A SIMPLE TECHNIQUE FOR SURFACE MODELING

MATTHEW BENG LAU

A SIMPLE TECHNIQUE FOR SURFACE MODELING

BY

© Matthew Beng Lau, B.Sc.

A thesis submitted to the School of Graduate
Studies in partial fulfillment of the
requirements for the degree of
Master of Science

Department of Computer Science
Memorial University of Newfoundland

March 1987

St. John's Newfoundland

ISBN 0-315-50444-7

ABSTRACT

Traditionally, a single unique curve or surface is generated by an algorithm for each set of control points, with Bezier and B-spline being two of the famous techniques used.

Recently, there has been increasing interest in the generation of many different curves or surfaces by a single technique for a set of control points. The Beta2-spline technique is a recent example of this idea. It uses a parameter called β_2 to act as a pulling agent, or tension, on the B-spline curve or surface. However, the curves or surfaces lose their smooth appearance as they are pulled closer to the set of control points. This could be an undesirable feature when smooth interpolation of the control points is desired.

The Bezier technique has been one of the most famous techniques in the computer aided design industry. It is capable of representing most of the geometric entities of practical interest. However, it has a shortcoming: the Bezier curve or surface bears little resemblance to the shape of the control polygon or the net respectively.

This paper proposes a simple technique which serves to overcome the two disadvantages mentioned above. The proposed technique employs a parameter called γ, to act as a pulling agent on the Bezier curve or surface. The pulled curves or surfaces have better mimicking properties than that of the Bezier. They always have a smooth appearance, even when interpolating the set of control points.

Acknowledgements

This thesis is dedicated to my Mom and Dad who have supported me throughout my academic years.

I would like to make special thanks to all those people who have helped me in my research and writing of my thesis, especially my supervisor, Professor R. James Dawe.

Special thanks also go to Dolores Campbell, Professor Jane Foltz, and Alan Tsui.

TABLE OF CONTENTS

LIST OF FIGURES

1. Prologue

1.1. Introduction

Surface Modeling techniques have undergone many improvements since their first introduction into computer graphics literature. Many techniques have been implemented in an effort to model surfaces. These techniques can be classified under the headings of interpolation and approximation.

Given a set of data points, interpolation is done by finding a function which passes through these data points. Some classic methods are Coon's method [9], Overhauser's [6,14], and the popular Lagrange and Hermit interpolation. On the other hand, approximation is concerned with finding a function which approximates the shape of the control vertices (i.e., data points), not necessarily passing through them. Two of the classical methods are the B-spline [15] and Bezier [4] methods.

The techniques mentioned in the references above generate a single unique representation based on the set of data points. In order to generate a surface which slightly deviates from the standard surface, either some data points have to be modified or another set of control points is needed. Recently, Brian Barsky proposed a method called the Beta-spline technique [1]. This is a generalization of the B-spline technique. It enables the generation of a set of different surfaces or curves just by varying the value of two parameters called Beta1 and Beta2. He also proposed a simpler method called the Beta2-spline [2], which employs just one parameter, Beta2, from the Beta-spline technique. This simplified method, a special case of Beta-spline , possesses characteristics that make the usage of Beta-2 spline very attractive. In fact, the idea of this paper is motivated by this special

technique. We will discuss this technique briefly in Chapter Four.

Analogous to the Beta2-spline, this paper proposes a technique that can generate different curves or surfaces for a single set of control vertices. This is done by varying the value of a single parameter which has been named **gamma**. The proposed method is simple. It is a modification of the Bezier and Overhauser techniques. The detail will be examined in Chapter Five.

The remainder of this chapter contains some fundamental mathematical concepts related to surface modeling. Chapter Two discusses the interpolation technique introduced by Overhauser [14]. In the third chapter, we talk about the Bezier technique [5] which has enjoyed popularity since its first implementation. Chapter Four discusses briefly the Beta2-spline technique, since it motivates the idea of this paper. The formulation of the proposed method is discussed in Chapter Five. Chapter Six describes an implementation of the proposed gamma-parameter technique on a SUN-Workstation. Chapter Seven concludes the thesis. The Appendix contains some of the pictures generated by the Gamma method.

1.2. Type Of Representation

A surface can be represented by either a non-parametric equation or by a parametric equation. Each representation has its own advantages.

1.2.1. Non-parametric Representation

Non-parametric surfaces are represented by an algebraic equation of the form :

$$F(x,y,z) = 0$$

In this form, it is easy to determine whether a point lies on the surface.

1.2.2. Parametric Representation

Parametric surfaces are generated by 3 bivariate functions of the form :

$$x = X(u,v)$$
$$y = Y(u,v)$$
$$z = Z(u,v)$$

In this form, we can easily generate all points on the surface by varying the parameters, u and v, appropriately.

1.3. Advantages Of Parametric Representation

For reasons which will be discussed later, the dominant representation of shapes in computer-aided design is the parametric form, that is, a 2-space curve is represented by a set of two functions

$$x = X(t)$$
$$y = Y(t)$$

of a parameter t. Hence, we can say that a point on such curve is represented by the vector

$$P_c = \left[X(t)\ Y(t)\right]$$

Likewise, a point on a space curve is given by the vector

$$P_c = \left[X(t)\ Y(t)\ Z(t)\right]$$

and a point on the surface is represented by the vector

$$P_s = \left[X(u,v)\ Y(u,v)\ Z(u,v)\right]$$

Reference [8] presents many reasons for using the parametric form of a curve vis-a-vis the non-parametric form. A few of these and some others are now discussed. Although our discussion below and the next section concerns general curves, the same

argument holds true for surfaces.

In general, the non-parametric curve form has several inherent drawbacks. A curve which is defined by tangent properties as well as points may well require that the slope be infinite. This can be avoided by either changing the coordinate axis or by using a different form of equation, both of which are cumbersome procedures. Curve segments must be bounded by defining the end points, but the test to determine whether a point lies on the bounded segment can be elaborate and even ambiguous when the curve loops. In the case of two dimensional curves, a given value of x may yield several values of y which must be tested, and in the case of twisted curves, the difficulty is compounded. This complicates the computation for displaying, plotting, etc., of points on the curve; computations may involve evaluating square, cube and higher powers. If the curve is to be plotted, either as a series of points or as a series of straight lines, the number of computations involved to generate a visually smooth curve would be very large. Parametric methods overcome many of these difficulties.

Consider the parametric representation of a general continuous curve in 3-space or a transformation of the form

$$\begin{aligned} x &= f(u) \\ y &= g(u) \\ z &= h(u) \end{aligned}$$

defined for u in the interval [a,b]. In vector notation

$$\bar{r}(u) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f(u) \\ g(u) \\ h(u) \end{bmatrix}$$

then

$$\frac{d\bar{r}}{du} = \begin{bmatrix} \frac{dx}{du} \\ \frac{dy}{du} \\ \frac{dz}{du} \end{bmatrix} = \begin{bmatrix} f'(u) \\ g'(u) \\ h'(u) \end{bmatrix}$$

is the tangent vector. The real slopes of the curve are given by the ratio of the components of the tangent vector. For example :

$$\frac{dy}{dx} = \frac{\left[\frac{dy}{du}\right]}{\left[\frac{dx}{du}\right]} = \frac{g'(u)}{f'(u)}$$

To specify an infinite slope, we need only to set one component of the tangent vector to zero.

A parametric curve is bounded by two parametric values. Each point on the curve corresponds to a unique parametric value. Thus, the test for a point lying on the curve reduces to finding the parametric value defining the point and checking that this value lies in the stated range. Parametric representation is axis independent and therefore a parametric curve is easily transformed into another of similar form by matrix multiplication. Similar transformations on a non-parametric curve are more difficult. A parametric curve can be subdivided into segments, each represented by a new set of parametric equations. We will discuss this in Chapter Three along with the Bezier technique. Last, but not least, in his PhD dissertation, Blinn [5] proposed a faster and more efficient method for displaying surfaces : the scanline method, which works well only with parametric surfaces.

Hence, the parametric form is not only more general, it is well suited to computation and display. In fact, the parametric form is the most convenient for graphics appli-

cations. Therefore it will be used in this thesis.

1.4. Objectives In Design

The ease of designing an object depends heavily on the characteristics of the technique used. Some of the most desirable characteristics are described below.

In the process of designing an object, one would like to be able to modify parts of the curve without affecting other parts. A technique has a local property if local modifications do not propagate. On the other hand, a change in the location or the multiplicity of one of the control points for a curve which has global property requires the recalculation of the whole curve, even though the change will have little effect far from the changed point. Piecewise polynomial functions offer a direct way of achieving local control. One such function is shown in Figure 1.1.

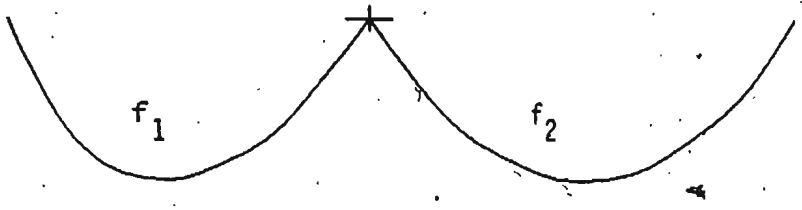


Figure 1.1 Two piecewise curves, f_1 and f_2, joined together

Generally, we wish to have a curve which is as smooth as possible. This implies that we need a higher order parametric curve. Unfortunately, higher order parametric curves tend to have undesirable wiggles or oscillations. For most purposes, a parametric cubic curve is sufficient. The cubic is also the lowest-order parametric which can describe a non-planar curve, a necessity for describing 3D curves. If the control points bound the curve then the curve exhibits the "convex hull" property. Intuitively, the convex hull of points in a plane is the area defined by a rubber band stretched around all the points.

The convex hull is useful in clipping a curve against a window or view volume. This is mentioned in [7]. We will talk more about the convex hull property when we discuss the Bezier curve. Usually, it is possible to convert a curve which does not have the convex hull property to one that does. For the reasons mentioned above, only the piecewise parametric cubic form is used in this thesis. Presented below is the general parametric cubic curve.

$$\begin{aligned} X(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ Y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\ Z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \end{aligned} \tag{1.1}$$

for t in the interval [0,1].

The derivative of X(t), Y(t), and Z(t) with respect to the parameter t are all of the same form. For example

$$\frac{dx}{dt} = 3a_x t^2 + 2b_x t + c_x \tag{1.2}$$

As mentioned in section 1.2, the three derivatives form the tangent vectors. The slopes of the curve are ratios of the tangent vector components. For example

$$\begin{aligned} \frac{dy}{dx} &= \frac{\frac{dy}{dt}}{\frac{dx}{dt}} \\ \frac{dz}{dx} &= \frac{\frac{dz}{dt}}{\frac{dx}{dt}} \\ \frac{dz}{dy} &= \frac{\frac{dz}{dt}}{\frac{dy}{dt}} \end{aligned} \tag{1.3}$$

Note that the slopes are independent of the lengths of the tangent vectors. If we multiply the derivatives by K, we have

$$K\frac{dx}{dt}$$
$$K\frac{dy}{dt}$$
$$K\frac{dz}{dt} \tag{1.4}$$

Then

$$\frac{K\frac{dy}{dt}}{K\frac{dx}{dt}} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{dy}{dx}$$

$$\frac{K\frac{dz}{dt}}{K\frac{dz}{dt}} = \frac{\frac{dz}{dt}}{\frac{dz}{dt}} = \frac{dz}{dz} \tag{1.5}$$

1.5. Parametric Cubic Curve Representation

We have seen the advantages that the parametric form has, now we are going to look at the general parametric form used to represent a curve. Reference [13] examines in detail the construction of different curves with different constraints. The next chapter will talk about the construction of curves and surfaces using Overhauser's technique. Chapter Three deals with the Bezier technique. A simple way to define a cubic parametric curve, the Hermit form, is considered here.

1.5.1. Hermit Form

The Hermit form of a cubic is determined from the endpoints and endpoint tangents. Since the derivations and resulting formulae for $Y(t)$ and $Z(t)$ are exactly analogous to that of $X(t)$, we will work explicitly with derivations only for $X(t)$, to find ways to define a_x, b_x, c_x, and d_x of (1.1). Given the points P_1 and P_4 and the tangent vectors R_1 and R_4 (where P_1 and P_4 refer to starting and ending points respectively,

and R_s and R_e refer to the starting and ending point tangents respectively), we can find a_x, b_x, c_x, and d_x for (1.1), subject to the conditions :

$$X(0) = P_{s_x}$$
$$X(1) = P_{e_x}$$
$$X'(0) = R_{s_x}$$
$$X'(1) = R_{e_x} \tag{1.6}$$

The subscript x is used to refer to the x-components of the points and the tangent vectors.

From (1.1), we can rewrite $X(t)$ as

$$X(t) = [t^3 \ t^2 \ t \ 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}_x \tag{1.7}$$

$$= [t^3 \ t^2 \ t \ 1] C_x \tag{1.8}$$

$$= TC_x \tag{1.9}$$

where T is a row vector of powers of t, and C_x is the column vector of coefficients of $X(t)$.

From (1.6), we get

$$X(0) = P_{s_x} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} C_x \tag{1.10}$$

$$X(1) = P_{e_x} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} C_x \tag{1.11}$$

In the case of the tangent vector conditions, we first differentiate (1.8) with respect to t, getting

$$X'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} C_x \tag{1.12}$$

Then

$$X'(0) = R_{0_x} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} C_x \tag{1.13}$$

$$X'(1) = R_{1_x} = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} C_x \tag{1.14}$$

The four conditions in (1.10), (1.11), (1.13) and (1.14) can be gathered together into a single matrix equation :

$$\begin{bmatrix} P_0 \\ P_1 \\ R_0 \\ R_1 \end{bmatrix}_x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} C_x \tag{1.15}$$

C_x can be solved by multiplying by the inverse of the 4 x 4 matrix throughout (1.15),

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ R_0 \\ R_1 \end{bmatrix}_x = M_h G_{h_x} \tag{1.16}$$

Here M_h is the Hermit matrix and G_h is the Hermit geometric vector. Substituting (1.16) into (1.9) we get

$$X(t) = TM_h G_{h_x} \tag{1.17}$$

By the same argument

$$Y(t) = TM_h G_{h_y} \tag{1.18}$$

$$Z(t) = TM_h G_{h_z} \tag{1.19}$$

The equation of a curve is frequently represented as

$$P(t) = TM_h G_h \tag{1.20}$$

Given P_s, P_e, R_s and R_e, we can evaluate $X(t)$, $Y(t)$ and $Z(t)$ for t in the interval |0, 1|, and find all points lying on the curve from P_s to P_e with starting tangent vector R_s and ending tangent vector R_e.

We can expand (1.20) and rearrange as

$$X(t) = P_{s_x} b_1 + P_{e_x} b_2 + R_{s_x} b_3 + R_{e_x} b_4 \tag{1.21}$$

where

$$\begin{aligned} b_1 &= 2t^3 - 3t^2 + 1 \\ b_2 &= -2t^3 + 3t^2 \\ b_3 &= t^3 - 2t^2 + t \\ b_4 &= t^3 - t^2 \end{aligned} \tag{1.22}$$

The four functions b_i $(1 \leq i \leq 4)$ in (1.22) are often called blending functions, since b_1 and b_2 blend P_s and P_e, while b_3 and b_4 blend R_s and R_e, producing the "blended" sum $X(t)$.

Obviously, the length of the tangent vector, and its direction, affects the appearance of the resulting curve. These are discussed more clearly in [7].

Different techniques have their own distinct set of blending functions. As can be seen from Chapter Three, the Bezier technique has blending functions

$$\begin{aligned} b_1 &= (1-t)^3 \\ b_2 &= 3t(1-t)^2 \\ b_3 &= 3t^2(1-t) \\ b_4 &= t^3 \end{aligned} \tag{1.23}$$

for a curve defined by four points P_1, P_2, P_3 and P_4 with t in the range [0, 1]. A drawback of the Hermit form is that it does not have the convex hull property. To be a convex hull, the set of blending functions must sum up to one for any value of the parameter t. Since all the blending functions in (1.23) sum up to one for any value of t in the

range [0, 1], the Bezier curve has the convex hull property. Another drawback of the Hermit form is that tangent vectors must be directly specified. This makes it difficult to draw, or to modify, a curve interactively. A novice might find it uncomfortable to use, since the concept of a tangent might be unfamiliar. However, forcing a curve to match a known tangent vector is easy with the Hermit form.

Nowadays, we frequently use a set of points to guide the drawing of a curve, called guiding points. For example, the Bezier technique uses four guiding points for each curve segment. This makes it a lot more convenient for interactive use. The Bezier curve is of the form

$$P(t) = TM_B G_i \tag{1.24}$$

We have the Hermit form as

$$TM_h G_i$$

and the Bezier form as

$$TM_B G_i$$

It is now possible to convert from one form to another; for example, Hermit form to Bezier form is done as follows

$$G_b = M_B^{-1} M_h G_i \tag{1.25}$$

Now a new set of control points G_i is found, which defines a bounded curve, congruent to that of Hermit, when the Bezier technique is applied to G_i. Thus, the Hermit form, which does not have the convex hull property, can be converted to the Bezier form, which does have this property.

1.6. Parametric Bicubic Surface Representation

In the preceding section, we examined a method, the Hermit method, to construct a curve given two points and their tangent values. In this section, we will talk about the

general principles of surface construction.

A curve is represented in 3-space by a vector-valued function

$$P(u) = \left[X(u)\ \ Y(u)\ \ Z(u)\right]$$

Generally, the curve-generating algorithm may be represented by an operator Φ_u applied to the vector-valued function $P(u)$ representing the data [10]

$$Q(u) = \Phi_u P(u)$$

As mentioned before, a point on a surface in 3-space can be described in parametric form by the function

$$P(u,v) = \left[X(u,v)\ \ Y(u,v)\ \ Z(u,v)\right]$$

Then the surface-generating procedure may be symbolically denoted by

$$Q(u,v) = \Phi_{u,v} P(u,v)$$

where $P(u,v)$ represents the data from which $Q(u,v)$ will be constructed. The most widely used approach to approximate $\Phi_{u,v}$ is to form the tensor product (alias cartesian product or cross product) of two univariate operators, leading to the surface approximation

$$Q(u,v) = \Phi_u \cdot \Phi_v P(u,v)$$

The effect of this operation is that Φ_v operates on the data $P(u,v_j)$ while Φ_u operates simultaneously on the data $P(u_i,v)$. Given a finite number of discrete values $P(u_i,v_j)$ of the function $P(u,v)$, we have

$$Q(u,v) = \sum_{j=0}^{n}\sum_{i=0}^{m} P(u_i,v_j)\cdot U_{i,m}(u)\cdot V_{j,n}(v) \qquad (1.26)$$

where $U_{i,m}(u)$ and $V_{j,n}(v)$ are interpolating, or approximating, univariate functions. Usually, U and V are of the same type, but this is not mandatory.

In matrix notation we may write (1.26) as

$$Q(s,t) = U \times P \times V^T \tag{1.27}$$

with

$$U = \begin{bmatrix} U_{0,m}(u) & U_{1,m}(u) & \cdots & U_{n,m}(u) \end{bmatrix}$$
$$V = \begin{bmatrix} V_{0,n}(v) & V_{1,n}(v) & \cdots & V_{n,n}(v) \end{bmatrix} \tag{1.28}$$

and P being the matrix of constraints

$$P = \begin{bmatrix} P(u_0,v_0) & \cdots & P(u_0,v_n) \\ P(u_1,v_0) & \cdots & P(u_1,v_n) \\ \vdots & \cdots & \vdots \\ P(u_m,v_0) & \cdots & P(u_m,v_n) \end{bmatrix} \tag{1.29}$$

The superscript T in (1.27) stands for the transpose of the matrix.

It follows naturally that the parametric equations representing a Hermit Surface are

$$X(s,t) = SM_h\, Q_x\, M_h^T T^T$$
$$Y(s,t) = SM_h\, Q_y\, M_h^T T^T$$
$$Z(s,t) = SM_h\, Q_z\, M_h^T T^T \tag{1.30}$$

It is shown in [7] that

$$Q_x = \begin{bmatrix} x_{00} & x_{01} & \frac{dx}{dt_{00}} & \frac{dx}{dt_{01}} \\ x_{10} & x_{11} & \frac{dx}{dt_{10}} & \frac{dx}{dt_{11}} \\ \frac{dx}{ds_{00}} & \frac{dx}{ds_{01}} & \frac{d^2x}{dsdt_{00}} & \frac{d^2x}{dsdt_{01}} \\ \frac{dx}{ds_{10}} & \frac{dx}{ds_{11}} & \frac{d^2x}{dsdt_{10}} & \frac{d^2x}{dsdt_{11}} \end{bmatrix} \tag{1.31}$$

Q_y and Q_z have the same form as (1.31) except that dx is replaced by dy and dz respectively.

Equation (1.30) defines a bicubic surface patch since the maximum power of either s or t is three. Note that partial derivatives are required for the computation of a Hermit Surface. This makes it cumbersome and impractical for a novice user. To overcome this problem, many techniques use guiding points or guiding planes to define the surface. That is, only points are specified in (1.31) which are required in the computation of the defined surface. Bezier and B-spline are two of the techniques which use guiding points or guiding planes. The proposed method in this thesis will use guiding points/planes as well.

1.7. Summary

In summary, we would like to recap some of the things that we have described so far. In computer-aided design today, the parametric form is used dominantly for reasons discussed in Section 1.3. To prepare for the later chapters, we have also discussed the general derivation of parametric cubic curves, the generalization of cubic curves to bicubic surfaces by tensor product, and the possibility of conversion between different forms of representations, for instance , from Hermit to Bezier.

2. Overhauser Technique

2.1. Introduction

In Chapter One, we looked briefly at a simple interpolation technique known as the Hermit Interpolation technique. In this chapter, a different but easier to use interpolation technique known as the Overhauser technique is discussed.

The Overhauser technique was originally proposed by A. W. Overhauser [14] in 1968 in an internal research report at Ford Motor Company. The method is based on the parabolic blending of curves and surfaces. A significant advantage of this technique is the user-oriented control of surface shape that is achieved because one interactively manipulates only coordinates on the design surface. This means that coordinate points which lie on the curve segments and surface patches make up the boundary condition, whereas in the Hermit method, parametric derivatives and control points are required to be specified to form a curve segment or a surface patch. In this thesis, the coordinate points, as used above, are treated in a way analogous to control points, since both of them are used to control the shape of the surfaces or curves.

2.2. Overhauser Curve

Since the derivation of the surface formulae can be generalized from that of the curve easily (for example, using cartesian product), we will work on the formulation of the curve first.

Each segment of the Overhauser curve is formulated from four adjacent points ; p_1, p_2, p_3 and p_4. As shown in Figure 2.1, p_1, p_2 and p_3 form a parabola $p(r)$ while p_2, p_3

and p_4 form a parabola $q(s)$. The Overhauser curve $c(t)$ which interpolates p_2 and p_3 is a blend of the above two parabolas, $p(r)$ and $q(s)$. Thus the Overhauser method can be expressed as:

$$c(t) = (1-t)p(r) + tq(s) \tag{2.1}$$

where r, s and t are parameters and c, p and q are vector expressions for curves.

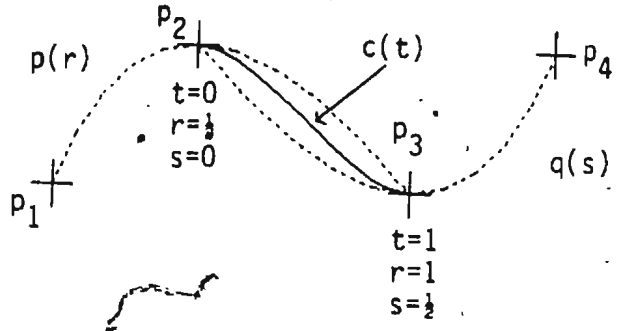


Figure 2.1 An Overhauser curve

Since p and q are parabolic curves, we can express them as second degree polynomials (parametric quadratic)

$$\begin{aligned} p(r) &= [r^2\ r\ 1]B \\ q(s) &= [s^2\ s\ 1]C \end{aligned} \tag{2.2}$$

where B and C are column matrices. By having r, s and t related in a linear manner,

$$\begin{aligned} r &= k_1 t + k_2 \\ s &= k_3 t + k_4 \end{aligned} \tag{2.3}$$

the resulting curve c (see Figure 2.1) is a parametric cubic :

$$c(t) = [t^3\ t^2\ t\ 1]A \tag{2.4}$$

Now, we need to express A in terms of the control points p_1, p_2, p_3 and p_4. By arbitrarily assigning parameter values at points p_1, p_2, p_3 and p_4, the relationship among parameters (eqs. (2.3)) may be specifically defined. Assuming,

$$p(0) = p_1, \quad p(\tfrac{1}{2}) = p_2, \quad p(1) = p_3$$

$$q(0) = p_2, \quad q(\tfrac{1}{2}) = p_3, \quad q(1) = p_4$$

$$c(0) = p_2, \quad c(1) = p_3 \tag{2.5}$$

We can get from eqs. (2.3)

$$r = \frac{1}{2}(t+1)$$

$$s = \frac{1}{2}t \tag{2.6}$$

From eq. (2.2)

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ \frac{1}{4} & \frac{1}{2} & 1 \\ 1 & 1 & 1 \end{bmatrix} B = \alpha B$$

we get

$$B = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{2.7}$$

Similarly, the matrix C of eq. (2.2) can be expressed in terms of p_2, p_3 and p_4 as

$$C = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_2 \\ p_3 \\ p_4 \end{bmatrix} \tag{2.8}$$

Substituting eqs. (2.2) into eq. (2.1) gives

$$c(t) = (1-t)\begin{bmatrix} r^2 & r & 1 \end{bmatrix} B + t\begin{bmatrix} s^2 & s & 1 \end{bmatrix} C \tag{2.9}$$

and eqs. (2.6) into eq. (2.9) we get

$$c(t) = \begin{bmatrix} \frac{-1}{4}(t^3+t^2-t-1) & \frac{-1}{2}(t^2-1) & -(t-1) \end{bmatrix} B + \begin{bmatrix} \frac{t^3}{4} & \frac{t^2}{2} & 1 \end{bmatrix} C \tag{2.10}$$

By substituting eq. (2.7) and eq. (2.8) into eq. (2.10),

$$c(t) = \begin{bmatrix} f_1 & f_2 & f_3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + \begin{bmatrix} g_1 & g_2 & g_3 \end{bmatrix} \begin{bmatrix} p_2 \\ p_3 \\ p_4 \end{bmatrix} \tag{2.11}$$

where

$$f_1 = \frac{-t^3}{2}+t^2-\frac{t}{2}, \quad f_2 = t^3-\frac{t^2}{2}-t+1, \quad f_3 = \frac{-t^3}{2}+\frac{t}{2}$$

and

$$g_1 = \frac{t^3}{2}-\frac{3}{2}t^2+t, \quad g_2 = -t^3+2t^2, \quad g_3 = \frac{t^3}{2}-\frac{t^2}{2} \tag{2.12}$$

By rearranging eq. (2.11)

$$c(t) = \begin{bmatrix} f_1 & f_2 & f_3 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} + \begin{bmatrix} 0 & g_1 & g_2 & g_3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

or

$$c(t) = \begin{bmatrix} f_1 & f_2+g_1 & f_3+g_2 & g_3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \tag{2.13}$$

By substituting eqs. (2.12) into eq. (2.13) we get

$$c(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} \frac{-1}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{1}{2} \\ 1 & \frac{-5}{2} & 2 & \frac{-1}{2} \\ \frac{-1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

$$= TM_G P \tag{2.14}$$

Therefore, in eq. (2.4) we have

$$A = \begin{bmatrix} \frac{-1}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{1}{2} \\ 1 & \frac{-5}{2} & 2 & \frac{-1}{2} \\ \frac{-1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \tag{2.15}$$

2.3. Characteristic of Overhauser Curve

Blending of two parabolas is possible only if the interval is an interior one. If the curve starts at point p_1 in Figure 2.1, then interpolation between p_1 and p_2 should be by the single parabola, defined in eq. (2.4), through p_1, p_2 and p_3. In the formulation above, we assign the parametric value uniformly between the control points as illustrated in eqs. (2.5). However, there is no requirement that the points be equally, or nearly equally spaced. Obviously, where higher curvature is needed, point density should be higher too.

The manner of construction (as a blend of two parabolas) guarantees that spurious wiggles will not be introduced as frequently happens when simple cubics are forced to pass through four points of a curve. The smoothness between adjacent curve segments is a very important factor since their application is usually to the design of some physical object such as an airplane wing. With the Overhauser method, the first derivative continuity is guaranteed between adjacent curve segments. The proof is shown below:

Suppose the curve starts at p_2 and control points start at p_1, as shown in Figure 2.2. The curve segment c_i is then computed from p_i, p_{i+1}, p_{i+2} and p_{i+3}. On the other hand, c_{i-1} is computed from p_{i-1}, p_i, p_{i+1} and p_{i+2}.

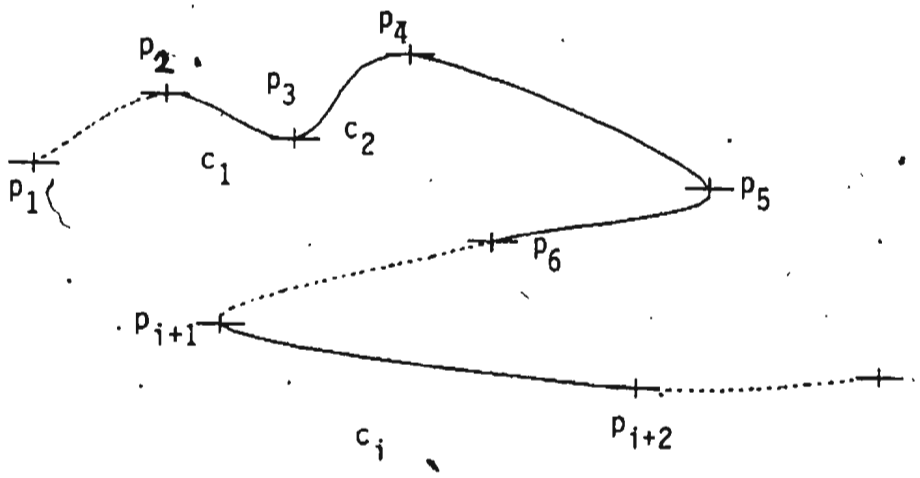


Figure 2.2 Piecewise Overhauser curves

From eq. (2.14), the first derivative of c_i is

$$c'_i(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{-1}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{1}{2} \\ 1 & \frac{-5}{2} & 2 & \frac{-1}{2} \\ \frac{-1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{bmatrix} \tag{2.16}$$

and the first derivative of c_{i-1} is

$$c'_{i-1}(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{-1}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{1}{2} \\ 1 & \frac{-5}{2} & 2 & \frac{-1}{2} \\ \frac{-1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix} \tag{2.17}$$

At the joint between c_{i-1} and c_i, the parametric value t for c_{i-1} is one, and the parametric value t for c_i i zero. That is

$$c_{i-1}(1) = c_i(0) \tag{2.18}$$

If the first derivative is continuous at the joint, then

$$c'_{i-1}(1) = c'_i(0)$$

To see this, we proceed as follows,

From eq. (2.16)

$$c'_i(0) = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{-1}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{1}{2} \\ 1 & \frac{-5}{2} & 2 & \frac{-1}{2} \\ \frac{-1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{bmatrix}$$

$$= \frac{-1}{2}p_i + \frac{1}{2}p_{i+2} \tag{2.19}$$

and from eq. (2.17)

$$c'_{i-1}(1) = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} \frac{-1}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{1}{2} \\ 1 & \frac{-5}{2} & 2 & \frac{-1}{2} \\ \frac{-1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix}$$

$$= \frac{-1}{2}p_i + \frac{1}{2}p_{i+2} \tag{2.20}$$

Since eq. (2.19) and eq. (2.20) agree with each other, the first derivative continuity is established at the joint of the curve segment c_{i-1} and c_i. To achieve a smoother curve, higher order of derivative continuity is needed. Unfortunately, it can be shown easily that the Overhauser method does not provide continuity at the joints automatically beyond the first derivative. However, for most purposes, first derivative continuity is sufficient.

Sometimes, a discontinuity in the first derivative is desired, for example at point p_4 of Figure 2.3. To achieve this, points off the blended curve, as indicated by p_4 and p_5

may be added.

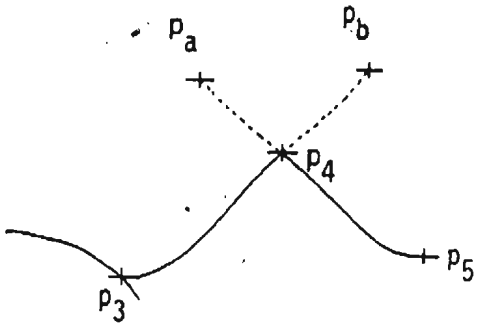


Figure 2.3 Forming a Discontinuity in First Derivative

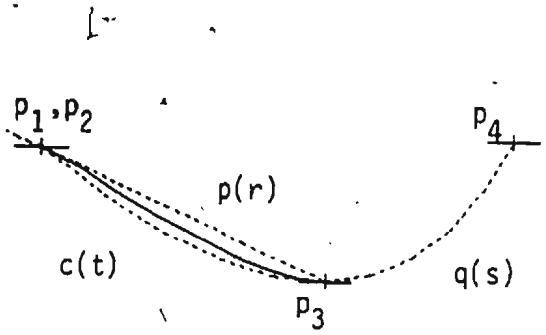


Figure 2.4 An Overhauser curve with two coincident points

It is possible to allow two points which define an Overhauser curve to be coincident as shown in Figure 2.4. This means that point p_a, point p_b or both points p_a and p_b of Figure 2.3 may coincide with point p_4. This yields first derivative discontinuity at point p_4 also. Consequently, end-points of a sequence of points may coincide with the second and second-to-last points. In this case, all points will lie on the curve.

Another important property that the Overhauser method possesses is that it has local control. Modification of point p_3 to p'_3 affects its four neighboring curve segments only as indicated in Figure 2.5. This property is desired because one can modify a part of the curve without affecting other parts of the curve.

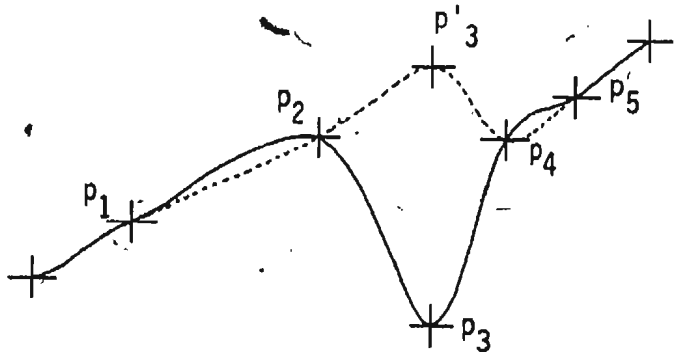


Figure 2.5 Modification of a point in an Overhauser curve

While the convex hull property is an important feature in computer graphics, the Overhauser method does not provide this facility. This is intuitively clear, since the curve has to pass through the control points and maintain smoothness (first derivative continuity) at the joint. A way to get around this is to convert the same curve segment to one that has the convex hull property. An example of this is to convert to the Bezier curve which was mentioned in Chapter One.

2.4. Overhauser Surfaces

There are many ways to construct a surface for a given mesh of points, for instance, by lofting or modified Coon technique. These are discussed in reference [14] by A. W. Overhauser. In this thesis, we will deal with the cartesian product or tensor product to form a surface patch.

From eq. (1.27), we have

$$Q(u,v) = U \times P \times V^T \tag{2.21}$$

By having $U = V$ in eq. (2.21), and from eq. (2.14), the Overhauser surface can be represented by

$$Q(s,t) = SM_O P M_O^T T^T \tag{2.22}$$

where S is a row of power s, T is a row of power t, and P is a 4 x 4 matrix containing sixteen control points. Therefore, a mesh of sixteen points is required to form an Overhauser surface patch as shown in Figure 2.6.

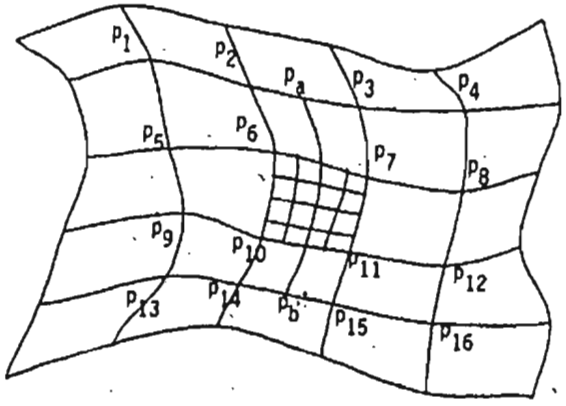


Figure 2.6 An Overhauser patch

The characteristics of the Overhauser curve also apply to the Overhauser surface. First derivative continuity is automatically achieved at the boundary of the adjacent surface patches. Local control property holds since a modification to a control point common to four patches affects twelve adjacent patches. Obviously, the convex hull property does not apply to the Overhauser surface nor does it apply to the Overhauser curve.

2.5. Summary

In this chapter, the derivation and some merits of the Overhauser method are discussed. The main purpose of this thesis is to derive a formula which will be done in Chapter Three, so that we can proceed with the proposed **gamma** surface easily in Chapter Five. For more detailed examination of the Overhauser method, please refer to reference [14].

3. Bezier Technique

3.1. Introduction

So far, we have considered the formulation of curves or surfaces to interpolate a given set of control points. Another approach, as mentioned in Chapter One, is to provide a good, smooth representation of a curve or a surface that approximates a given set of control points. A very famous approximation technique is proposed by P. Bezier, which we will be looking at in this chapter. P. Bezier works for the French firm Regie Renault. His UNISURF system had been used by designers to define objects such as the outer panels of cars and yacht hulls [4] since 1972.

3.2. Bezier Curve

Let V_i $(i = 0, 1, \cdots, m)$ be ordered control points in R^k, the Bezier polygon is formed by joining successive points $P = V_0 V_1 \cdots V_m$. Then the Bezier curve $B(t)$ associated with the Bezier polygon is given by

$$B(t) = \sum_{i=0}^{m} V_i \Phi_{i,m}(t) \tag{3.1}$$

for $0 \leq t \leq 1$, where

$$\Phi_{i,m}(t) = \binom{m}{i} t^i (1-t)^{m-i} \tag{3.2}$$

are the blending functions called the Bernstein polynomials.

Eq. (3.1) is a vector valued equation, thus parametric functions for X, Y and Z coordinates can be written as

$$X(t) = \sum_{i=0}^{m} X_i \Phi_{i,m}(t)$$

$$Y(t) = \sum_{i=0}^{m} Y_i \Phi_{i,m}(t)$$

$$Z(t) = \sum_{i=0}^{m} Z_i \Phi_{i,m}(t) \tag{3.3}$$

for $0 \leq t \leq 1$, where (X_i, Y_i, Z_i) constitute the control point V_i.

From eq. (3.1) and eq. (3.2), we get

$$B(0) = V_0$$

and

$$B(1) = V_m \tag{3.4}$$

Therefore, a Bezier curve interpolates its endpoints. Figure 3.1 shows a cubic Bezier curve with endpoints interpolated.

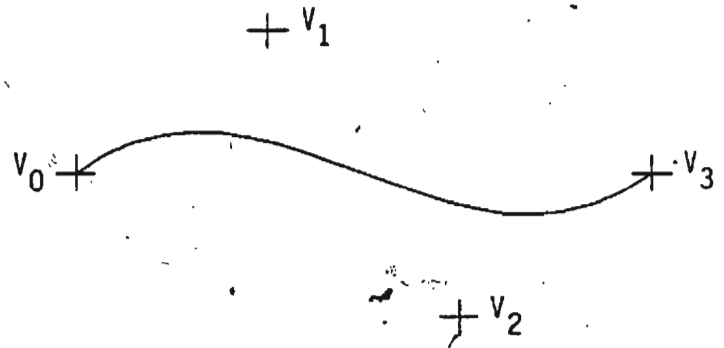


Figure 3.1 A cubic Bezier curve

3.3. Convex Hull

By using the Binomial Theorem, it is easy to show that a Bezier curve lies within the convex hull of its defining control points. This is shown below:

The blending functions as shown in eq. (3.2) are evidently nonnegative on |0,1|. From the Binomial Expansion Theorem, we have

$$\sum_{i=0}^{m} \binom{m}{i} t^i (1-t)^{m-i} = \frac{(t-0+1-t)^m}{(1-0)^m} = 1 \tag{3.5}$$

Since the blending functions sum to one and are nonnegative on |0,1|, the Bezier curve

must lie within the convex hull of the defining control points.

3.4. Tangent or First Derivative Continuity

Many segments of the Bezier curve can be joined together. In order to make two curve segments have a smooth transition at the joint, we proceed as follows :

Differentiating eq. (3.1) we see that

$$B'(0) = m(V_1 - V_0) \tag{3.6}$$

and

$$B'(1) = m(V_m - V_{m-1}) \tag{3.7}$$

so that tangents at either end are collinear with the line segment between the first two and the last two control points respectively. Consecutive segments in a composite Bezier curve (see Figure 3.2) can therefore be made tangent continuous simply by arranging that the penultimate control point of the first curve, V_2, the shared endpoint, $V_3 = V'_0$, and the second vertex, V'_1 of the next curve be collinear. If $|V_3 - V_2| = |V'_1 - V'_0|$, then the composite curve has first derivative continuity. To achieve higher order continuity, more complicated work has to be done. This is covered in [4].

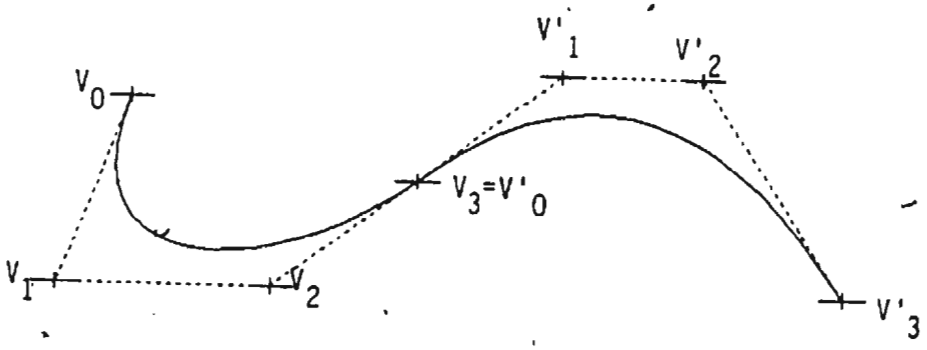


Figure 3.2 Two cubic Bezier curves with tangent continuity at the joint

3.5. Cubic Bezier Curve

A simple Bezier curve such as the one defined by eq. (3.1) is a single polynomial, and is therefore infinitely differentiable. However, adding control points increases the degree of the curve as well as increasing the cost of evaluation. Also, the movement of any control point alters the Bezier curve entirely. This obviously is not a pleasant feature to possess. An alternative, as suggested in Chapter One, is to construct a composite Bezier curve by having the last control point of the i^{th} segment and the first control point of the $(i+1)^{st}$ segment coincide. In this thesis, we will work with the cubic Bezier curve only. That is, each Bezier curve segment is defined by four control points. One example is shown in Figure 3.1. Each cubic Bezier curve is of the form

$$B(t) = V_0(1-t)^3 + V_1 3t(1-t)^2 + V_2 3t(1-t) + V_3 t^3 \tag{3.8}$$

or in matrix form

$$B(t) = [t^3\ t^2\ t\ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix} \tag{3.9}$$

$$= TM_B V \tag{3.10}$$

Figure 3.3 shows the four blending functions that correspond to a cubic Bezier curve with four control points.

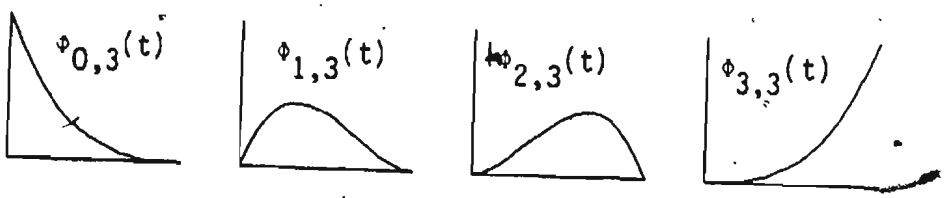


Figure 3.3 The four cubic Bezier blending functions

These curves represent the "influence" that each control point exerts on the curve for various values of t. When $t=0$, the first control point, V_0, corresponding to $\Phi_{0,3}(t)$ is most influential; in fact, locations of all other control points are ignored when $t=0$, because their blending functions are zero. When $t=1$, the situation is symmetric for V_3. The middle control points V_1 and V_2 are most influential when $t=\frac{1}{3}$ and $\frac{2}{3}$ respectively.

3.6. Local Control

Local control can be achieved easily with a piecewise cubic Bezier curve. If the control point modified is an interior one, then only one Bezier segment needs to be modified. If, however, the control point modified is at the joint of two Bezier curve segments, two Bezier segments need to be modified.

3.7. Subdivision of Cubic Bezier Curve

As mentioned in Section 1.3, a parametric curve can be subdivided into segments, each represented by a new set of equations. We would like to find an easy way to break a cubic Bezier curve in half. That is, suppose we have the Bezier curve (see Figure 3.4)

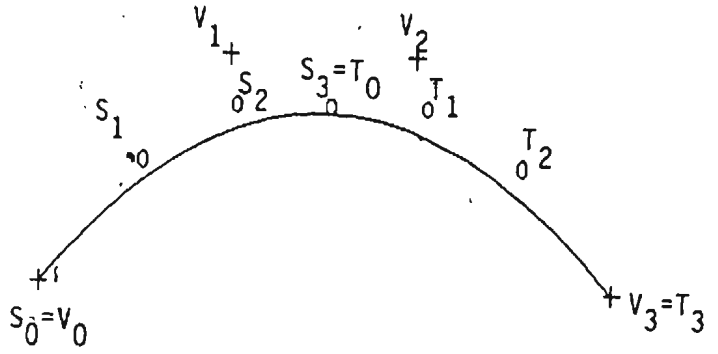


Figure 3.4 Subdivision of a cubic Bezier curve. The original control vertices V_0, V_1, V_2, V_3 are represented with '+'s. The new control vertices are represented with '0's

$$B(t) = V_0(1-t)^3 + V_1 3t(1-t)^2 + V_2 3t^2(1-t) + V_3 t^3 \tag{3.11}$$

We need to find control points

$$S_0 \;\; S_1 \;\; S_2 \;\; S_3 \;\; T_0 \;\; T_1 \;\; T_2 \;\; T_3 \tag{3.12}$$

such that the Bezier curve

$$L(s) = S_0(1-s)^3 + S_1 3s(1-s)^2 + S_2 3s^2(1-s) + S_3 s^3 \tag{3.13}$$

for $0 \leq s \leq 1$ is the first half of the curve defined by V_0, V_1, V_2 and V_3 (i.e. $B(t)$ for $0 \leq t \leq 0.5$) and

$$R(u) = T_0(1-u)^3 + T_1 3u(1-u)^2 + T_2 3u^2(1-u) + T_3 u^3 \tag{3.14}$$

for $0 \leq u \leq 1$ is the second half of the curve defined by V_0, V_1, V_2 and V_3 (i.e. $B(t)$ for $0.5 \leq t \leq 1.0$)

To find the control points listed in (3.12), we proceed as follows :

We know that

$$S_0 = V_0 \tag{3.15}$$

$$S_3 = B(\frac{1}{2}) = \frac{1}{8}(V_0+3V_1+3V_2+V_3) \tag{3.16}$$

From eq. (3.6) and eq. (3.7) we know that

$$L'(0) = 3(S_1-S_0) \tag{3.17}$$

$$L'(1) = 3(S_3-S_2) \tag{3.18}$$

Since we have $t = \frac{s}{2}$, by the chain rule

$$L'(0) = \frac{1}{2}B'(0) = \frac{3}{2}(V_1-V_0) \tag{3.19}$$

and

$$L'(1) = \frac{1}{2}B'(\frac{1}{2}) = \frac{3}{8}(V_3+V_2-V_1-V_0) \tag{3.20}$$

Now, from eq. (3.15) to eq. (3.20), we have

$$3(S_1-S_0) = \frac{3}{2}(V_1-V_0)$$

$$3(S_3-S_2) = \frac{3}{8}(V_3+V_2-V_1-V_0)$$
$$S_3 = \frac{1}{8}(V_0+3V_1+3V_2+V_3)$$
$$S_0 = V_0 \tag{3.21}$$

Solving eqs. (3.21) above, we get

$$S_0 = V_0$$
$$S_1 = \frac{1}{2}(V_0+V_1)$$
$$S_2 = \frac{1}{4}(V_0+2V_1+V_2)$$
$$S_3 = \frac{1}{8}(V_0+3V_1+3V_2+V_3) \tag{3.22}$$

Analogously, we can show that

$$T_0 = \frac{1}{8}(V_0+3V_1+3V_2+V_3)$$
$$T_1 = \frac{1}{4}(V_1+2V_2+V_3)$$
$$T_2 = \frac{1}{2}(V_2+V_3)$$
$$T_3 = V_3 \tag{3.23}$$

The fact that Bezier curves and surfaces have the convex hull and subdivisibility properties is extremely useful in computer graphics. In [12], Lane and Riesenfeld proposed an efficient algorithm to generate curves and surfaces, and an efficient algorithm to find the intersection between curves or surfaces based on the subdivision and convex hull properties. Because a Bezier curve lies within the convex hull of its defining control points, we can test to see if the length of the convex hull is within some tolerance of the distance between the first and the last control points [11], or whether the distance between each pair of control points is less than some tolerance, or whether the deviation of internal control vertices from a line segment joining the end points is sufficiently small [12], etc. The convergence test can be applied to each subdivided curve individually, so

that the subdivision process ceases appropriately when the curve has become "locally flat", at which point drawing the control polygon directly is a sufficiently good approximation to the real curve. The convex hull property can also be used for finding the intersection points between two curves. Since each curve segment lies within the convex hull of its defining polygon, two curves do not intersect if their convex hulls do not intersect. If the convex hulls do intersect, we can continue to subdivide the curves and test for the possibility of convex hulls intersection between all appropriate subdivided curve segments. When two convex hulls are sufficiently small, yet still overlap, we can declare that a point of intersection has been found. Reference [7] mentioned that the convex hull property is useful in clipping a curve against a window or view volume. We clip the curve itself only if the intersection between the convex hull of the curve and the window or the view volume is detected first.

In this thesis, we only deal with subdivision at the midpoint of a cubic Bezier curve. A general technique for directly subdividing elsewhere other than at the midpoint can be found in [3]

3.8. Bezer Bicubic Surfaces

The formulation of the three dimensional Bezier surface can be easily derived from the formulation of the Bezier curve by using the cartesian product or tensor product of two curves. Two similar blending functions are used, one for each parameter:

$$B(s,t) = \sum_{i=0}^{n} \sum_{j=0}^{m} V_{i,j}\, \Phi_{i,n}(s)\Phi_{j,m}(t) \tag{3.24}$$

For a bicubic surface, $m = n = 3$ in eq. (3.24). Eq. (3.24) can also be expressed in matrix form:

$$B(s,t) = SM_B\, VM_B^T T^T \tag{3.25}$$

where S is a row of power s, T is a row of power t, and V is a 4 x 4 matrix containing sixteen control points. Therefore, as in the Overhauser case, a mesh of sixteen points is required to form a Bezier bicubic surface patch as shown in Figure 3.5.

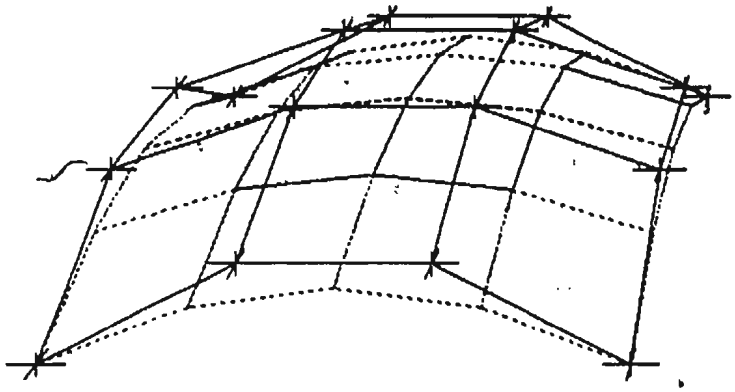


Figure 3.5 A bicubic Bezier surface patch. The net is represented by the solid lines, and the dotted lines outline the surface

We say a net is formed when each control point in the given mesh is connected to all its adjacent control points by straight lines. Analogus to the Bezier curve, the Bezier surface is contained within the convex hull of its net.

The extension of curves to surfaces leaves the properties listed in previous sections unchanged. Surfaces can be pieced together from individual patches as shown in Figure 3.6.

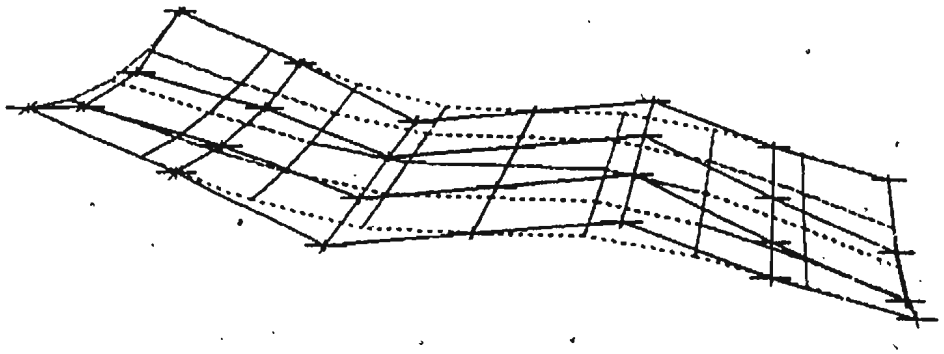


Figure 3.6 Two bicubic Bezier surfaces pieced together

Achieving continuity between two patches can be done much the same way as that at the joint between two curves. Continuity across patch boundaries is obtained by making the four control points on the boundary equal. Achieving first-order continuity across a boundary requires not only that first and last edges of the joining polygons be collinear, but also that the ratio of the lengths of all these edge pairs be constant. Of course, as in the Bezier curve, the Bezier surface has the convex hull property, and can be subdivided into smaller sub patches too. Local control can be achieved easily with the bicubic Bezier patches. If the control point modified is interior to the patch, then a patch needs to be modified. However, if the modified point is on the boundary of two patches, or on the corner of four patches, then two, or four patches need to be modified respectively.

3.9. Conclusion

The Bezier technique has been widely used for the following reasons: the intuitive feel of the control points, the control of continuity, the ability to subdivide and having the convex hull property. It is well suited to interactive design, since the control points can be easily manipulated to change the shape of the surface.

Although the Bezier technique possesses many benefits as stated above, it does have shortcomings. The Bezier technique discussed above generates a unique single curve or a unique single surface representation to its control polygon or its net, respectively. A shortcoming of the Bezier method is that the curve and the surface often bear little resemblance to the shape of the polygon and the net. An example is shown in Figure 3.7. In Figure 3.7, no loop is produced by the Bezier algorithm where we expected one to exist. Thus, it would be nice if we can generate a curve or surface that can mimic the shape of the control polygon or the net more closely than that generated by the technique mentioned above. That is the purpose of this thesis.

In Chapter Five, we propose a method, such that by varying the value of a parameter named **gamma** , the curve or surface can mimic the control polygon or the net closely, or even interpolate the set of control points, while maintaining its smooth appearance.

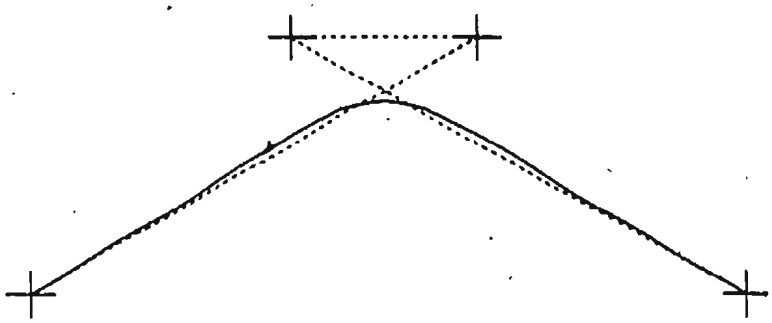


Figure 3.7 A bezier curve with no loop where one is expected

4. Beta2-spline Technique

As mentioned in Chapter One, the idea of this thesis is motivated by the Beta2-spline technique. Thus, before going into detail of our proposal, we would like to discuss briefly the Beta2-spline technique.

4.1. Introduction

A spline curve or surface is a piecewise function with continuity constraints at the locations where the pieces of the function meet (often called the joints in the case of curves, and borders or boundaries in the case of surfaces). Undoubtedly, the Overhauser and Bezier techniques, as discussed in previous chapters, are special cases of the spline technique.

B-spline is another special spline technique. It is very popular in the CAD/CAM industry today. It has local control, the convex hull property and achieves C^1 and C^2 automatically across the curve or surface. It generates a single unique representation to the set of control points. It is shown in [1] that the generated curve or surface may or may not interpolate any of the control points, depending on the multiplicity of the knots and the multiplicity of the control points. As the multiplicity of the knots or the control points increases, the continuity at the joint or at the border decreases.

A generalization of the B-spline method proposed by Barsky [2] is called the Beta2-spline method. As in the B-spline technique, Beta2-spline technique has local control properties in the sense that modifying a control vertex will affect the shape of the curve or the surface locally. It has global properties also in the sense that by modifying a parameter called Beta2, β_2, the whole curve or surface can be modified to mimic the control

polygon or the net more closely, or even to interpolate the control points. It has the convex hull property. No matter what value β_2 has, the curve or the surface always lies in the convex hull of the control polygon or the net, respectively. The Beta2-spline imposes the first derivative continuity automatically. The curvature vector continuity at the joint or the border is controlled by the value of the parameter, β_2.

4.2. Beta2-spline Formula

For reference purposes, the formula for the Beta2-spline is produced below. The complete formulation is shown in [2].

The i^{th} segment of a Beta2-spline curve takes the form

$$Q_i(\beta_2 ; u) = \sum_{r=-2}^{1} V_{i+r}\, b_r(\beta_2 ; u); \qquad 0 \leq u \leq 1 \tag{4.1}$$

The four functions $b_i(\beta_2 ; u)$ $(-2 =< i \leq 1)$ are the blending functions or the basis functions for the Beta2-spline curve.

It is shown in [2] that

$$\begin{aligned}
b_{-2}(\beta_2 ; u) &= 2\gamma(1-u)^3 \\
b_{-1}(\beta_2 ; u) &= \gamma(\beta_2+8+u^2(-3(\beta_2+4)+2u(\beta_2+3))) \\
b_0(\beta_2 ; u) &= \gamma(2+u(6+u(3(\beta_2+2)-2u(\beta_2+3)))) \\
b_{-1}(\beta_2 ; u) &= 2\gamma u^3
\end{aligned} \tag{4.2}$$

where

$$\gamma = \frac{1}{\beta_2+12} \tag{4.3}$$

Note that the basis functions of eqs. (4.2) are symmetric in the sense that $b_{-2}(\beta_2 ; u) = b_1(\beta_2 ; 1-u)$ and $b_{-1}(\beta_2 ; u) = b_0(\beta_2 ; 1-u)$. This symmetry implies that reversing the order of the control vertices in a control polygon reverses the resulting curve.

Eqs. (4.1) can be written in matrix form

$$Q_i(\beta_2 ; u) = [u^3 \; u^2 \; u \; 1] M_\beta \begin{bmatrix} V_{i-2} \\ V_{i-1} \\ V_i \\ V_{i+1} \end{bmatrix} \tag{4.4}$$

where

$$M_\beta = \gamma \begin{bmatrix} 2 & \beta_2+8 & 2 & 0 \\ -6 & 0 & 6 & 0 \\ 6 & -3(\beta_2+4) & 3(\beta_2+2) & 0 \\ -2 & 2(\beta_2+3) & -2(\beta_2+3) & 2 \end{bmatrix} \tag{4.5}$$

and where γ is specified in eq. (4.3)

Naturally, the Beta2-spline surface can be formed by taking the tensor product of two curves

$$Q_{i,j}(\beta_2 ; u \; ; t) = UM_\beta V M_\beta^T T^T \tag{4.6}$$

where U and T are vectors of power u and t respectively, and V is a 4 x 4 matrix containing 16 control points required to form a surface patch.

4.3. Characteristics

By varying the parameter, β_2, a different curve or surface can be generated. When $\beta_2 = 0$, the spline has continuity of first and second derivative vectors; it is thus equivalent to a cubic uniform B-spline when $\beta_2 = 0$. As β_2 increases, the Beta2-spline curve or surface is pulled closer to the polygon or the net. This suggests that β_2 acts like a tension parameter. In the limit of infinite β_2, the Beta2-spline curve becomes a piecewise linear function that interpolates the interior vertices of the control polygon. In the case of a surface, in the limit of infinite β_2, the surface coincides with the interior panels

of the net. This makes the curve or the surface lose its smooth appearance as the value of the β_2 goes to infinity. Figure 4.1 and Figure 4.2 show the effect of β_2 on the Beta2-spline curve and surface respectively.

4.4. Summary

The Beta2-spline is a generalization of the B-spline method. It can generate a curve or surface closer to, or even interpolate the bounding polygon or the net. However, as the curve or the surface interpolates the polygon or the net, the smoothness at the joint or the border is lost due to the necessity of satisfying the convex hull property. This technique is useful for some modeling as suggested in [2]. However, for some applications, when smooth interpolation is needed, this technique is not appropriate.

In the next chapter, a method is proposed to generalize the Bezier and Overhauser techniques instead of the B-spline technique. The proposed technique uses a parameter named **gamma** to act as a tension on the Bezier curve or surface. The newly generated curve or surface always maintains its smooth appearance, even if it interpolates the set of control points.

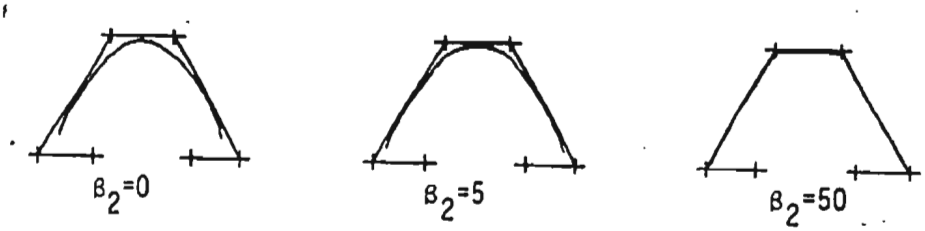


Figure 4.1 The curves above are generated with the Beta2-spline method for the same set of control vertices, with different parameter values

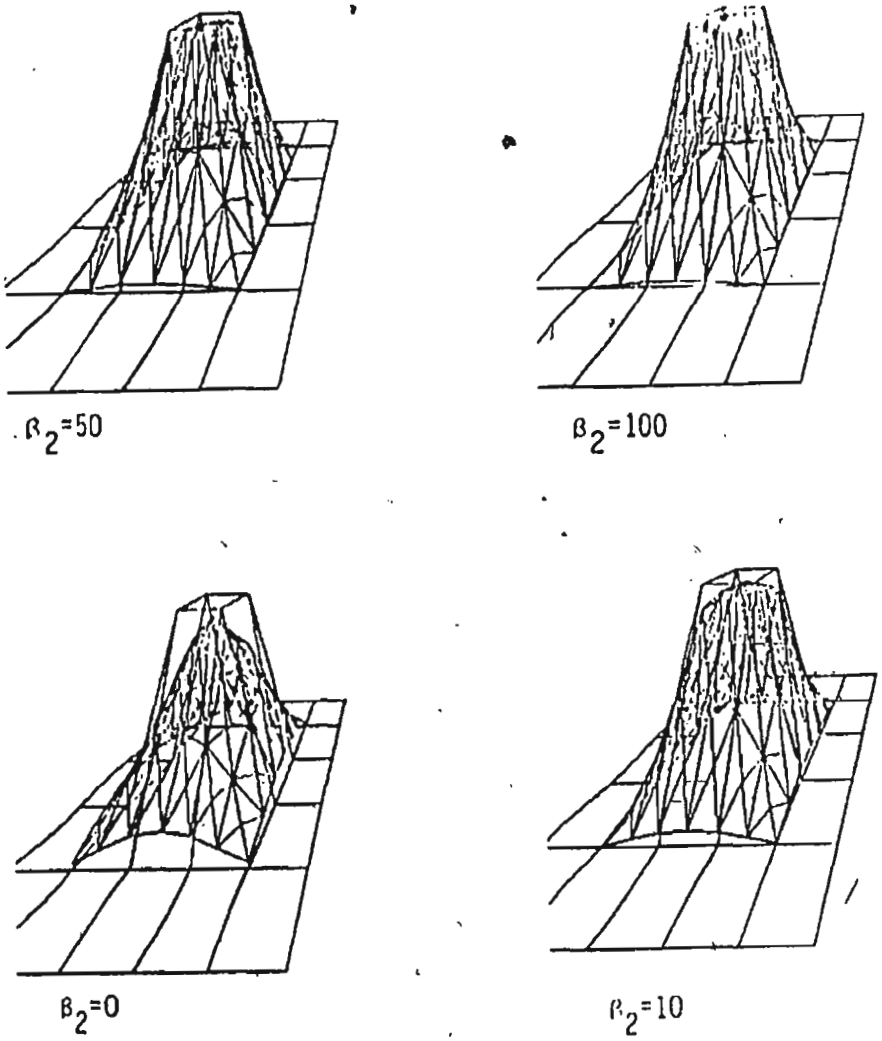


Figure 4.2 The surfaces above are generated with the Beta2-spline method for the same set of control vertices, with different parameter values

5. Gamma Technique

5.1. Introduction

In Chapter Four, a generalization of the B-spline method, the Beta2-spline technique, was briefly discussed. The Beta2-spline technique has the capability of generating a curve or a surface, which grows closer and closer to the control polygon or the net, simply by increasing the value of a parameter named β_2. Since the generated curve or surface must lie within the convex hull of the control polygon or the net, each curve segment or surface patch grows "straighter" as they grow closer to the control polygon or the net. Thus, the smooth appearance of the curve or the surface is lost as the value of β_2 increases.

In this thesis, a generalization to the Bezier and the Overhauser techniques, rather than the B-spline technique, is proposed. By increasing the value of the parameter, which has been named **gamma** , γ, from zero to one, the corresponding curve or surface can be made to grow closer to the control polygon or the net. A distinction with this method, as opposed to the Beta2-spline technique, is that the resulting curve or surface always maintains its smooth appearance even when it interpolates the control polygon or the net. However, by maintaining the smoothness, the convex hull property, as exhibited by the Bezier technique, is lost with the proposed technique. A way to get around this deficiency is to convert the gamma curve or surface to a Bezier curve or surface, which has the convex hull property as mentioned in Chapter One. This will be discussed again later.

5.2. Gamma Curve

Figure 5.1 shows three gamma curves with the same control polygon. The only

difference is that the left curve has $\gamma = 0.0$, the middle curve has $\gamma = 0.5$, and the right curve has $\gamma = 1.0$. Notice that the curve with $\gamma = 1.0$ interpolates the control polygon while maintaining the smoothness of the curve.

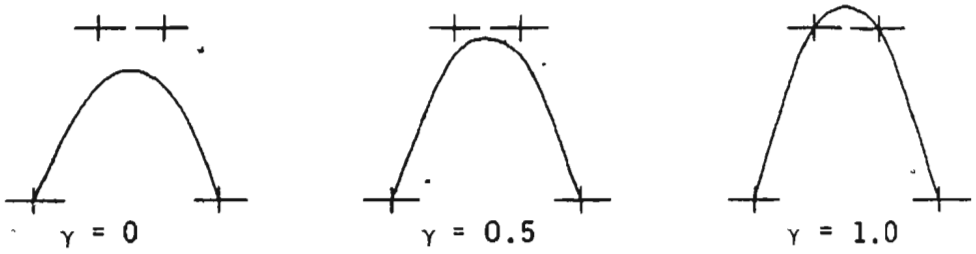


Figure 5.1 Gamma curves with the same set of control points, but different parameter values

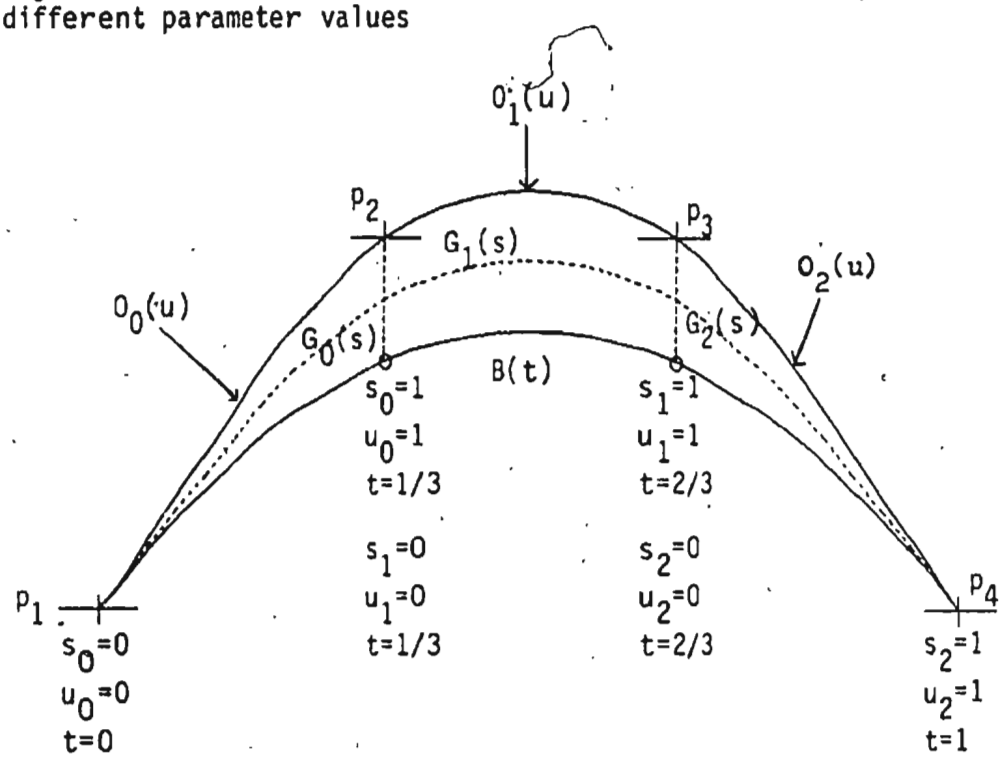


Figure 5.2 A Gamma curve

The following is an explanation of the formulation of a Gamma curve.

Suppose we are given six control vertices, p_0, p_1, p_2, p_3, p_4, and p_5. Then a segment of a cubic Bezier curve $B(t)$ can be drawn for the four interior control vertices, and three segments of an Overhauser curve $O_i(u)$ $(0 \leq i \leq 2)$ can be drawn to interpolate the interior four control vertices. This is shown in Figure 5.2. The gamma curve is represented by the three curve segments $G_0(\gamma ; s)$, $G_1(\gamma ; s)$, and $G_2(\gamma ; s)$ shown in Figure 5.2. Each gamma curve segment can be expressed as follows:

$$G_i(\gamma ; s) = (1-\gamma)B(t)+\gamma O_i(u) \tag{5.1}$$

for $(0 \leq \gamma \leq 1)$, and $(0 \leq i \leq 2)$.

From eq. (5.1), we can tell that the Gamma curve lies on the Bezier curve, $B(t)$, when $\gamma = 0$, and on the Overhauser curve, O_i, when $\gamma = 1$. Any other value of γ will generate a curve that is in between the Bezier and the Overhauser curve. Thus, the Gamma technique is a generalization of the Bezier and the Overhauser techniques.

To derive the formulation, we assume that a cubic Bezier curve segment, combined with three consecutive Overhauser curve segments, is used to make up three consecutive Gamma curve segments. By arbitrarily partitioning the cubic Bezier curve into three parts, we can use the first part, combined with O_0, to derive G_0; the second part, combined with O_1, to derive G_1; and the third part, combined with O_2 to derive G_2.

By having s, t, and u of eq. (5.1) related in a linear manner

$$t = k_1 s + k_2$$

and

$$u = k_3 s + k_4 \tag{5.2}$$

The resulting curve $G_i(\gamma ; s)$ is a parametric cubic curve

$$(s^3 \; s^2 \; s \; 1)A \tag{5.3}$$

where A is a matrix to be found.

To find eqs. (5.2), we proceed as follows:

For $G_0(\gamma\,;\,s)$, we have

$$G_0(\gamma\,;\,0) = O_0(0) = B(0) = p_1 \tag{5.4}$$

When $\gamma = 1$, we have

$$G_0(1\,;\,1) = O_0(1) = p_2 \tag{5.5}$$

and when $\gamma = 0$ in eq. (5.1), we let

$$G(0\,;\,1) = B\left(\frac{1}{3}\right) \tag{5.6}$$

As a result from eq. (5.4), we have

$$\begin{aligned} s &= 0 \\ u &= 0 \\ t &= 0 \end{aligned} \tag{5.7}$$

and from eq. (5.5) and eq. (5.6)

$$\begin{aligned} s &= 1 \\ u &= 1 \\ t &= \frac{1}{3} \end{aligned} \tag{5.8}$$

Solving eq. (5.2) with constraints given in eqs. (5.7), and eqs. (5.8) we get

$$\begin{aligned} t &= \frac{s}{3} \\ u &= s \end{aligned} \tag{5.9}$$

Now, by substituting eqs. (5.9) into eq. (5.1) we get

$$G_0(\gamma\,;\,s) = (1-\gamma)B\left(\frac{s}{3}\right)+\gamma O_0(s) \tag{5.10}$$

Analogously, for segment $G_1(\gamma\,;\,s)$, we have

$$\begin{aligned} s &= 0 \\ u &= 0 \end{aligned}$$

$$t = \frac{1}{3} \tag{5.11}$$

and

$$\begin{aligned} s &= 1 \\ u &= 1 \\ t &= \frac{2}{3} \end{aligned} \tag{5.12}$$

In this case, we assume

$$G_1(1\,;\,0) = O_1(0) = p_2 \quad \text{and} \quad G_1(0\,;\,0) = B(\tfrac{1}{3})$$

$$G_1(1\,;\,1) = O_1(1) = p_3 \quad \text{and} \quad G_1(0\,;\,1) = B(\tfrac{2}{3})$$

when $s = 1$, $t = \frac{2}{3}$, since $B(t)$ is most affected by point P_3 when $t = \frac{2}{3}$. Solving eq. (5.2) with constraints given in eqs. (5.11) and eqs. (5.12), we get

$$\begin{aligned} t &= \frac{s+1}{3} \\ u &= s \end{aligned} \tag{5.13}$$

By substituting eqs. (5.13) into eq. (5.1), we have

$$G_1(\gamma\,;\,s) = (1-\gamma)B(\tfrac{s+1}{3}) + \gamma O_0(s) \tag{5.14}$$

Similarly, for segment $G_2(\gamma\,;\,s)$, we have

$$\begin{aligned} s &= 0 \\ u &= 0 \\ t &= \frac{2}{3} \end{aligned} \tag{5.15}$$

and

$$\begin{aligned} s &= 1 \\ u &= 1 \\ t &= 1 \end{aligned} \tag{5.16}$$

In this case, we assume

$$G_2(1\,;0) = O_2(0) = p_3 \quad \text{and} \quad G_2(0\,;0) = B(\frac{2}{3})$$

$$G_2(\gamma\,;1) = O_1(1) = B(1) = p_4$$

Now, by solving eq. (5.2) giving the constraints in eqs. (5.15) and eqs. (5.16) we get

$$i = \frac{s+2}{3}$$

$$s = u \tag{5.17}$$

Again, by substituting eqs. (5.17) into eq. (5.1) gives

$$G_2(\gamma\,;s) = (1-\gamma)B(\frac{s+2}{3})+\gamma O_2(s) \tag{5.18}$$

Now, we have the three equations as follows:

$$G_0(\gamma\,;s) = (1-\gamma)B(\frac{s}{3})+\gamma O_0(s)$$

$$G_1(\gamma\,;s) = (1-\gamma)B(\frac{s+1}{3})+\gamma O_1(s)$$

$$G_2(\gamma\,;s) = (1-\gamma)B(\frac{s+2}{3})+\gamma O_2(s) \tag{5.19}$$

Note that six control points p_1, p_2, p_3, p_4, p_5, and p_6 are needed for eqs. (5.19) as shown above. From Figure 5.2, we can see that the Bezier curve only depends on the four interior control points (p_1, p_2, p_3,p_4) for all three equations in eqs. (5.19). However, the first Overhauser curve segment O_0 depends on (p_1, p_2, p_3, p_4), the second Overhauser curve segment O_1 depends on (p_2, p_3, p_4, p_5), and the third Overhauser curve segment O_2 depends on (p_3, p_4, p_5, p_6). Therefore, we need six control points (p_1, p_2, p_3, p_4, p_5, p_6) to compute eqs. (5.19). If there are more than three segments, that is if more than six control points are given, then the general equation can be derived easily as

$$G_i(\gamma\,;s) = (1-\gamma)B_\sigma(\frac{s+\beta}{3})+\gamma O_i(s) \tag{5.20}$$

where

$$\sigma = i \,/\, 3$$

$$\beta = i \,\%\, 3 \tag{5.21}$$

The '/' sign in eq. (5.21) stands for the integer division of i divided by 3, and '%' sign stands for the integer remainder of i divided by 3.

The three Gamma curve segments shown in eqs. (5.19) form a Gamma curve that is in between the Bezier curve, $B(t)$ which is computed from $(p_1\ p_2\ p_3\ p_4)$, and the Overhauser curve which interpolates $(p_1\ p_2\ p_3\ p_4)$. As a result, the Gamma curve (consisting of three consecutive Gamma curve segments) will interpolate the control points p_1, and p_4 for any value of γ.

5.3. Simplification

Now that the general form of the curve equation has been found as given in eq. (5.20), a simplification can be made to derive an equation of the form given by eq. (5.3).

To do the simplification, we have to work with the three equations together as given in eqs. (5.19). Given a list of consecutive control points p_i $(0 \le i \le n$ where $n \ge 5$ and $(n+1) \,\%\, 3 = 0)$, there must be at least six control points and the number of control points must be divisible by three. These conditions must be satisfied because we need to have at least four control points (p_1, p_2, p_3, p_4) for a cubic Bezier curve segment, and we need two extra control points (p_0, p_5), one on each end of the four control points (p_1, p_2, p_3, p_4) so that the Overhauser curve can interpolate the four control points (p_1, p_2, p_3, p_4). Furthermore, if there is more than one Bezier curve segment, say m segments, then we need to have $3m+1$ control points for the m cubic Bezier curve segments to join together. For an Overhauser curve to interpolate these $3m+1$ control points, we need to have two exterior control points, one on each end of the list of the $3m+1$ control points. Consequently, $(3m+1)+2 = n$ control points are needed for a gamma curve to be built. Since $n = 3m+3$, and m must be an integer, n must be divisible by three.

However, since the control points are numbered from zero, the condition becomes $n+1$ must be divisible by three, and n must be greater than five.

The layout for a list of twelve, i.e. $n = 11$, control points is given in Figure 5.3.

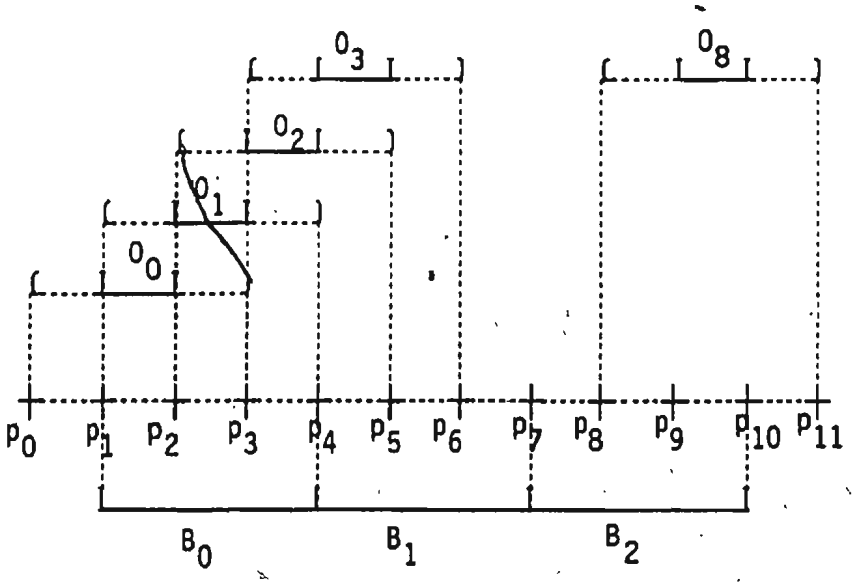


Figure 5.3 . The dependency of the Bezier and Overhauser curve on the set of the control points. 'O' stands for Overhaser curve and 'B' stands for Bezier curve

From Figure 5.3, we can see that nine Overhauser curve segments can be constructed with dependency on the control points as shown below:

$$O_0 \ \Omega \ (p_0\, p_1\, p_2\, p_3)$$
$$O_1 \ \Omega \ (p_1\, p_2\, p_3\, p_4)$$
$$O_2 \ \Omega \ (p_2\, p_3\, p_4\, p_5)$$
$$O_3 \ \Omega \ (p_3\, p_4\, p_5\, p_6)$$

$$O_8 \ \Omega \ (p_8\, p_9\, p_{10}\, p_{11}) \tag{5.22}$$

where Ω stands for 'depends on'. In general

$$O_i \ \Omega \ (p_i\ p_{i+1}\, p_{i+2}\, p_{i+3}) \tag{5.23}$$

Similarly, three Bezier curve segments can be constructed with control point dependency given as follows:

$$\begin{aligned} B_0 \ &\Omega \ (p_1\, p_2\, p_3\, p_4) \\ B_1 \ &\Omega \ (p_4\, p_5\, p_6\, p_7) \\ B_2 \ &\Omega \ (p_7\, p_8\, p_9\, p_{10}) \end{aligned} \tag{5.24}$$

In general

$$B_j \ \Omega \ (p_{3j+1}\, p_{3j+2}\, p_{3j+3}\, p_{3j+4}) \tag{5.25}$$

From Figure 5.3 and eqs. (5.19), we can see that in order to derive the Gamma curve, we must have

$$G_0 \ \Omega \ \left\{O_0\ B_0\right\} \ \Omega \ (p_0\, p_1\, p_2\, p_3\, p_4)$$

$$G_1 \ \Omega \ \left\{O_1\ B_0\right\} \ \Omega \ (p_1\, p_2\, p_3\, p_4)$$

$$G_2 \ \Omega \ \left\{O_2\ B_0\right\} \ \Omega \ (p_1\, p_2\, p_3\, p_4\, p_5)$$

$$G_6 \ \Omega \ \left\{O_6\ B_2\right\} \ \Omega \ (p_6\, p_7\, p_8\, p_9\, p_{10})$$

$$G_7 \ \Omega \ \left\{O_7\ B_2\right\} \ \Omega \ (p_7\, p_8\, p_9\, p_{10})$$

$$G_2 \;\Omega\; \left\{ O_2 \;\; B_2 \right\} \;\Omega\; (p_7\, p_8\, p_9\, p_{10}\, p_{11}) \tag{5.26}$$

In general

$$G_i \;\Omega\; \begin{cases} (p_i \;\; p_{i+1}\, p_{i+2}\, p_{i+3}\, p_{i+4}) & \text{if } i \,\%\, 3 = 0 \\ (p_i \;\; p_{i+1}\, p_{i+2}\, p_{i+3}) & \text{if } i \,\%\, 3 = 1 \\ (p_{i-1}\, p_i \;\; p_{i+1}\, p_{i+2}\, p_{i+3}) & \text{if } i \,\%\, 3 = 2 \end{cases} \tag{5.27}$$

The dependency of the first three Gamma segments on the set of control points is shown in Figure 5.4.

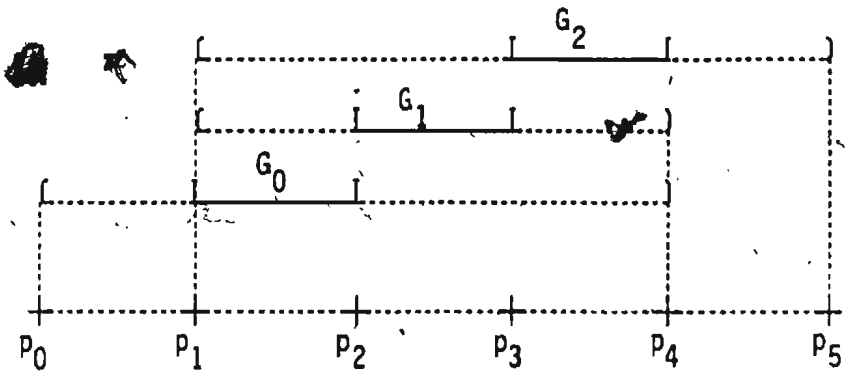


Figure 5.4 The dependency of the first three consecutive Gamma curve segments on the set of the control points

This also means that a total of six consecutive control points are needed to compute the three Gamma curve segments as indicated in eqs. (5.27). By taking the union of the control points in the eqs. (5.27), the general equation dependency on the control points becomes:

$$G_i \;\Omega\; \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \\ p_{i+5} \end{bmatrix} \tag{5.28}$$

where $\delta = i - (i \% 3)$. And since $\beta = i \% 3$ as shown in eq. (5.21), δ can be expressed as :

$$\delta = i - \beta \tag{5.29}$$

If $\beta = 0$, then the Gamma curve G_i will interpolate the control point p_{i+1} for any value of γ, and if $\beta = 2$, the Gamma curve G_i will interpolate the control point p_{i+4} for any value of γ.

One has to remember that not all six control points in eq. (5.28) are used to derive the *ith* Gamma curve segment. The set of control points used by the *ith* Gamma curve segment depends on the β value. When $\beta = 0$, the top five control points are needed for the computation of the *ith* Gamma curve segment. When $\beta = 1$, the interior four control points are needed, and when $\beta = 2$ the bottom five control points are required for the computation of the *ith* Gamma curve segment. Note that, for any value of i for the *ith* Gamma curve segment, the Bezier curve segment (used to define the *ith* Gamma curve segment) is always defined by the four interior control points. On the other hand, the set of control points used to define the Overhauser curve segment (which is used to define the i Gamma curve segment) always depend on the value of β. If $\beta = 0$, the top four control points are used. This is the reason for using the top five control points to derive the Gamma curve segment when $\beta = 0$ (the top four control points are used for Overhauser, and the middle four control points are used for Bezier, implying that a union of five control points is used to derive the Gamma curve segment with $\beta = 0$). If $\beta = 1$, the middle four are used. In this case, since both the Overhauser curve and the Bezier curve use the middle four control points, the Gamma curve segment uses the middle four control points for derivation when $\beta = 1$. And if $\beta = 2$, the bottom four control points are used. With the same argument as in the case when $\beta = 0$, four middle control points

for the Bezier curve and four bottom control points for the Overhauser curve, rendering a union of five control points for the derivation of the Gamma curve segment when $\beta = 2$. As a result of these, eq. (5.20) can also be written as:

$$G_i(\gamma\,;\,s) = (1-\gamma)TM_B \begin{bmatrix} p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \end{bmatrix} + \gamma UM_{O\_G} \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \\ p_{i+5} \end{bmatrix} \tag{5.30}$$

where T, U are row vectors of power of $t = \frac{s+\beta}{3}$, and $u = s$ respectively, and M_B is the matrix given in Chapter Three for the cubic Bezier curve, and M_{O_G} is a 4 x 6 matrix to be determined. Eq. (5.30) can be simplified into

$$G_i(\gamma\,;\,s) = ((1-\gamma)TM_{BT\_G} + \gamma UM_{O\_G}) \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \\ p_{i+5} \end{bmatrix} \tag{5.31}$$

where M_{BT_G} is a 4 x 6 matrix to be found.

It is easy to derive the matrix M_{BT_G}. From Chapter Three, we know the cubic Bezier curve matrix M_B is as follows :

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{5.32}$$

Now, since the matrix M_B only works on the four interior control points ($p_{i+1}\ p_{i+2}\ p_{i+3}\ p_{i+4}$), as given in eq. (5.30), we can write

$$M_{BT\_G} = \begin{bmatrix} 0 & -1 & 3 & -3 & 1 & 0 \\ 0 & 3 & -6 & 3 & 0 & 0 \\ 0 & -3 & 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.33}$$

so that the matrix M_{BT_G} would not affect the two exterior control points (p_i p_{i+3}) as required. Now, by substituting eq. (5.33) and $t = \frac{s+\beta}{3}$ into TM_{BT_G} of eq. (5.31), we get

$$[s^3 \; s^2 \; s \; 1]M_{B\_G} = TM_{BT\_G} \tag{5.34}$$

where

$$M_{B\_G} = \begin{bmatrix} 0 & \frac{-1}{27} & \frac{1}{9} & \frac{-1}{9} & \frac{1}{27} & 0 \\ 0 & \frac{1}{9}(3-\beta) & \frac{1}{3}(\beta-2) & \frac{1}{3}(1-\beta) & \frac{\beta}{9} & 0 \\ 0 & \frac{-1}{9}(\beta-3)^2 & \frac{1}{3}(\beta^2-4\beta+3) & \frac{1}{3}\beta(2-\beta) & \frac{\beta^2}{9} & 0 \\ 0 & \frac{-1}{27}(\beta-3)^3 & \frac{1}{9}\beta(\beta-3)^2 & \frac{1}{9}\beta^2(3-\beta) & \frac{\beta^3}{27} & 0 \end{bmatrix} \tag{5.35}$$

As a result, eq. (5.31) can also be written as

$$G_i(\gamma\,;s) = ((1-\gamma)SM_{B\_G} + \gamma UM_{O\_G}) \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \\ p_{i+5} \end{bmatrix} \tag{5.36}$$

where M_{B_G} is given in eq. (5.35), and S is a row of power of s.

To derive the matrix M_{O_G}, we proceed as follows :

From Chapter Two, we know that the Overhauser matrix is

$$M_O = \begin{bmatrix} \frac{-1}{2} & \frac{3}{2} & \frac{-3}{2} & \frac{1}{2} \\ 1 & \frac{5}{2} & 2 & \frac{-1}{2} \\ \frac{-1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{5.37}$$

Now, by assigning C_i $(0 \leq i \leq 3)$ to the *ith* column of the matrix M_O given in eq. (5.37), we can write

$$M_{O\_G} = \begin{bmatrix} C_0\alpha_{0,\beta} \\ C_1\alpha_{0,\beta}+C_0\alpha_{1,\beta} \\ C_2\alpha_{0,\beta}+C_1\alpha_{1,\beta}+C_0\alpha_{2,\beta} \\ C_3\alpha_{0,\beta}+C_2\alpha_{1,\beta}+C_1\alpha_{2,\beta} \\ C_3\alpha_{1,\beta}+C_2\alpha_{2,\beta} \\ C_3\alpha_{2,\beta} \end{bmatrix}^T \tag{5.38}$$

where the superscript T stands for the transpose of the matrix M_{O_G}, $\alpha_{i,\beta}$ is a kronecker delta, and $\beta = i \ \% \ 3$ as given in eq. (5.21). That is

$$\alpha_{i,\beta} = \begin{cases} 1 & \text{if } i = \beta \\ 0 & otherwise \end{cases} \tag{5.39}$$

In effect, the matrix M_{O_G} contains the matrix M_O which is being shifted β columns to the right. This is required because if $\beta = 0$, $\beta = 1$ and $\beta = 2$, we want only the top four, middle four, and the bottom four control points given in eq. (5.36) to form the Overhauser curve segment respectively for the construction of the *ith* Gamma curve segment. Consequently, at any moment, there are four non zero columns in the matrix M_{O_G} and the remaining two columns consist of zeros.

Since we know $u = s$, we can rewrite eq. (5.36) as

$$G_i(\gamma\,;s) = ((1-\gamma)SM_{B\_G}+\gamma SM_{O\_G}) \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \\ p_{i+5} \end{bmatrix} \tag{5.40}$$

which can be simplified into

$$G_1(\gamma\,;\,s) = [s^3\ s^2\ s\ \ 1]M_G P_v \tag{5.41}$$

where

$$M_G = (1-\gamma)M_{B\_G} + \gamma M_{O\_G} \tag{5.42}$$

and

$$P_v = \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \\ p_{i+5} \end{bmatrix} \tag{5.43}$$

Consequently, the matrix A in eq. (5.3) can be expressed as:

$$A = M_G P_v \tag{5.44}$$

The 4 x 6 matrix M_G is produced below:

$$M_G = \begin{bmatrix} -\frac{\gamma}{2}\alpha_0 & -\frac{(1-\gamma)}{27}+\frac{\gamma}{2}(3\alpha_0-\alpha_1) & \frac{(1-\gamma)}{9}+\frac{\gamma}{2}(-3\alpha_0+3\alpha_1-\alpha_2) \\ \gamma\alpha_0 & \frac{(1-\gamma)}{9}(3-\beta)+\frac{\gamma}{2}(2\alpha_1-5\alpha_0) & \frac{(1-\gamma)}{3}(\beta-3)+\frac{\gamma}{2}(4\alpha_0-5\alpha_1+2\alpha_2) \\ -\frac{\gamma}{2}\alpha_0 & -\frac{(1-\gamma)}{9}(\beta-3)^2-\frac{\gamma}{2}\alpha_1 & \frac{(1-\gamma)}{3}(\beta^2-4\beta+3)+\frac{\gamma}{2}(\alpha_0-\alpha_2) \\ 0 & -\frac{(1-\gamma)}{27}(\beta-3)^3+\gamma\alpha_0 & \frac{(1-\gamma)}{9}\beta(\beta-3)^2+\gamma\alpha_1 \end{bmatrix}$$

$$\begin{bmatrix} -\frac{(1-\gamma)}{9}+\frac{\gamma}{2}(\alpha_0-3\alpha_1+3\alpha_2) & \frac{(1-\gamma)}{27}+\frac{\gamma}{2}(\alpha_1-3\alpha_2) & \frac{\gamma}{2}\alpha_2 \\ \frac{(1-\gamma)}{3}(1-\beta)+\frac{\gamma}{2}(-\beta_0+4\alpha_1-5\alpha_2) & \frac{(1-\gamma)}{9}\beta+\frac{\gamma}{2}(-\alpha_1+4\alpha_2) & -\frac{\gamma}{2}\alpha_2 \\ \frac{(1-\gamma)}{3}\beta(2-\beta)+\frac{\gamma}{2}\alpha_1 & \frac{(1-\gamma)}{9}\beta^2+\frac{\gamma}{2}\alpha_2 & 0 \\ \frac{(1-\gamma)}{9}\beta^2(3-\beta)+\gamma\alpha_2 & \frac{(1-\gamma)}{27}\beta^3 & 0 \end{bmatrix} \tag{5.45}$$

In the above equation, each $\alpha_i \equiv \alpha_{i,\beta}$ where β is dropped for lack of space to produce the matrix.

Once again, the general equation for a Gamma curve is

$$G_i(\gamma\,;s) = [s^3\ s^2\ s\ 1]M_G \begin{bmatrix} p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \\ p_{i+4} \\ p_{i+5} \end{bmatrix} \tag{5.46}$$

where

$$\begin{aligned} \delta &= i - (i\ \%\ 3) \\ &= i - \beta \end{aligned}$$

The parameter γ gives the interpolated curve between the Bezier and the control points, or rather the Overhauser curve, and the parameter s determines the value along the interpolated curve. The matrix M_G requires the input of the value of γ, and the number, i, of the curve segment for the computation of β. Further simplification of the matrix M_G is discussed in the next Chapter (where an implementation of the Gamma technique on the Sun workstation is considered) by breaking the matrix into three simple cases.

5.4. Continuity

The Gamma curve will have positional and C^1 continuity, if the Bezier curve has positional and C^1 continuity. Since the Overhauser curve always has C^1 continuity as discussed in Chapter Two, the proof is trivial as shown below.

Suppose we want to prove that two adjacent Gamma curves G_i, and G_{i+1} have C^0, and C^1 continuity at the joint marked X as shown in Figure 5.5. Further we assume that the two adjacent Bezier curves $(B_j\ B_{j+1})$, and two adjacent Overhauser curves $(O_i\ O_{i+1})$ used to form the two Gamma curves have positional and C^1 continuity at the

joint X.

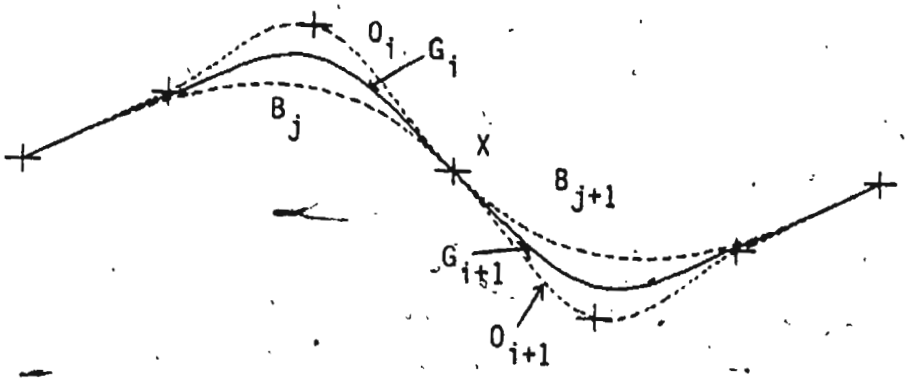


Figure 5.5 Two Gamma curves have first degree derivative continuity at the joint if the Bezier and the Overhauser curves used to derive them are c^1 continuous

Then at the joint X, we have

$$B_j(1) = B_{j+1}(0)$$
$$B'_j(1) = B'_{j+1}(0) \tag{5.47}$$

and

$$O_i(1) = O_{i+1}(0)$$
$$O'_i(1) = O'_{i+1}(0) \tag{5.48}$$

and

$$B_j(1) = O_{(1)} = B_{j+1}(0) = O_{i+1}(0) \tag{5.49}$$

Note that the formula for the Gamma curves G_i and G_{i+1} from the Figure 5.5 are

$$G_i(\gamma\,;s) = (1-\gamma)B_j(\frac{s+2}{3})+\gamma O_i(s)$$
$$G_{i+1}(\gamma\,;s) = (1-\gamma)B_{j+1}(\frac{s}{3})+\gamma O_{i+1}(s) \tag{5.50}$$

To prove positional continuity, we need to prove

$$G_i(\gamma\,;s) = G_{i+1}(\gamma\,;0) \tag{5.51}$$

The proof is as follows:

$$G_i(\gamma\,;\,1) = (1-\gamma)B_j(1)+\gamma O_i(1)$$
$$G_{i+1}(\gamma\,;\,0) = (1-\gamma)B_{j+1}(0)+\gamma O_{i+1}(0) \tag{5.52}$$

Now, by conditions specified by eqs. (5.49), we have

$$G_i(\gamma\,;\,s) = G_{i+1}(\gamma\,;\,0)$$

which is just what we wanted to prove.

To prove C^1 continuity, we need to prove

$$G'_i(\gamma\,;\,1) = G'_{i+1}(\gamma\,;\,0) \tag{5.53}$$

The proof is as follows:

Using the chain rule applied to eqs. (5.50), we get

$$G'_i(\gamma\,;\,1) = \frac{1}{3}(1-\gamma)B'_j(1)+\gamma O'_i(1)$$
$$G'_{i+1}(\gamma\,;\,0) = \frac{1}{3}(1-\gamma)B'_{j+1}(0)+\gamma O_{i+1}(0) \tag{5.54}$$

By conditions given in eqs. (5.47) and eqs. (5.48), we proved eq. (5.53), that is

$$G'_i(\gamma\,;\,1) = G'_{i+1}(\gamma\,;\,0)$$

Although a joint, X, is proven above to be C^0 and C^1 continuous, the proof applies to any other joints, as long as the Bezier curve is positional and C^1 continuous. If however the Bezier curve has tangent continuity, instead of C^1 continuity, the Gamma curve generated is guaranteed to have tangent continuity at the joint too. A very interesting feature that a Gamma curve has is that a curve that is not smooth can be made into a smooth curve as it interpolates the control polygon. Suppose at $\gamma = 0$, the curve generated, which is also a Bezier curve, does not have C^1 or tangent continuity, then as gamma increases from zero to one, the curve becomes smoother and smoother. As a matter of fact, when $\gamma = 1$, the curve generated has C^1 continuity. This is because when $\gamma = 1$, the corresponding curve generated is the Overhauser curve, which has C^1

continuity across the curve unless some points are duplicated. Figure 5.6 illustrates this case.

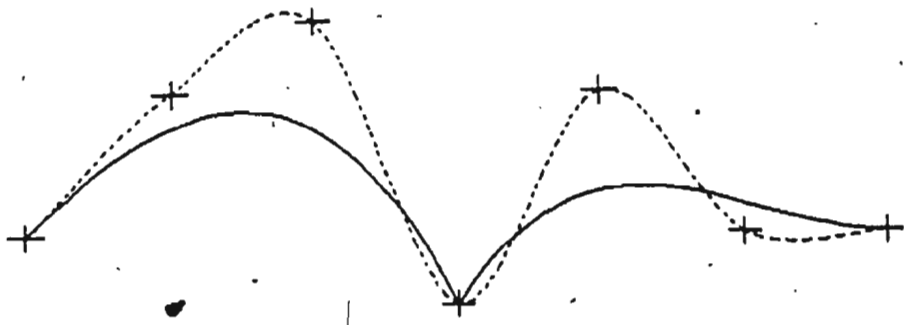


Figure 5.6 The solid curve has first derivative discontinuity with γ=0. The dotted curve has first derivative continuity with γ=1.0

5.5. Characteristics of γ

When $\gamma = 0$, the Gamma curve is analogous to the Bezier curve. As the value of γ increases towards one, the Gamma curve comes closer and closer to the set of control points. This suggests that γ can be used as a pulling agent on the Bezier curve towards the set of control points. When there is more than one piecewise cubic Bezier curve segment, with C^1 continuity maintained throughout the curve, the pulled curve always has a smooth appearance and is C^1 continuous. We mentioned in Chapter Three that a drawback of the Bezier technique is that the curve generated does not resemble the control polygon very much. However with the Gamma technique, the mimicking of the control polygon can easily be done just by increasing the value of the γ. A very striking result of this is shown in Figure 5.7, where an expected loop is obvious only when the value of γ is increased to 0.4. When $\gamma = 1$, the Gamma curve is the Overhauser curve, which interpolates the set of control points and maintains the first derivative continuity throughout. Due to the necessity of maintaining smooth interpolation, the convex hull property is

sacrificed. This is shown in Figure 5.8, where the solid line enclosed the convex hull of the control polygon and the dotted line represents the Gamma curve with $\gamma = 1$.

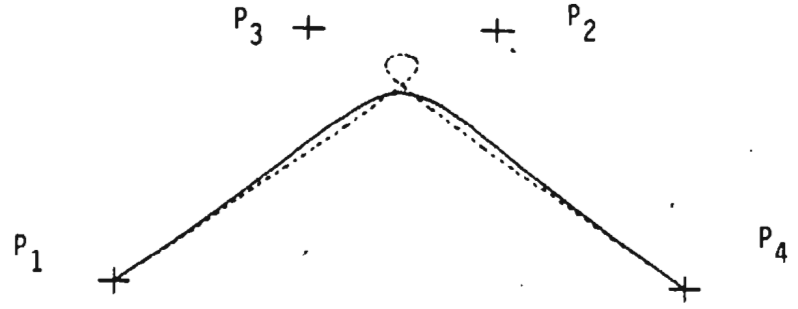


Figure 5.7 A loop occurs at γ=0.4

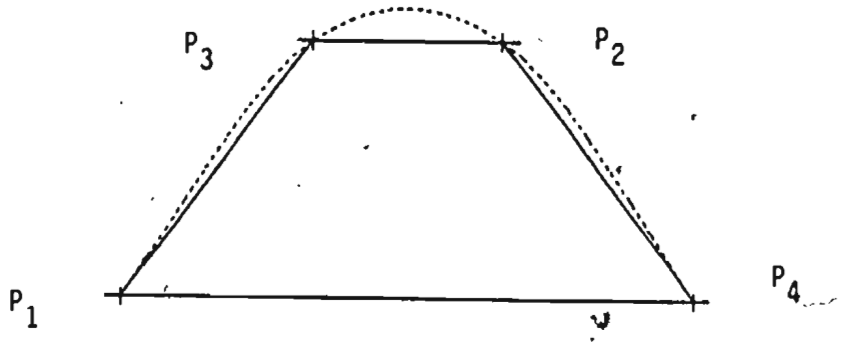


Figure 5.8 Gamma curve is not contained within the convex hull of the control polygon. Dotted line is the Gamma curve with γ=1.0 and the solid line encloses the convex hull of the control polygon

This undesirable property can be eliminated by converting the Gamma curve into the Bezier curve, whereby the gamma curve is constrained with a new set of control points.

5.6. Local Control

With the Gamma method, local control can be achieved. That is to say, by modifying a control point, only a few local curve segments need to be adjusted.

The number of Gamma curve segments which need to be adjusted depends solely on the position of where the modified control point lies. Assume that the Gamma curve

starts from the second control point and ends at the second-to-last control point (see Figure 5.9).

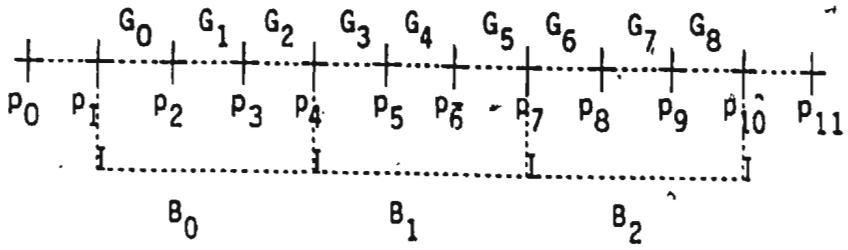


Figure 5.9 Layout of the relationship between the Gamma curve and the Bezier curve on the set of control points

Obviously, if either of the endpoints is changed, only one end Gamma curve segment derived from the modified control endpoint needs to be adjusted. If the modified control point is located at the joint of two Bezier cubic segments, then the two Bezier segments need to be adjusted, which of course implies that six consecutive Gamma curve segments need to be adjusted. For example, if the control point P_7 is modified in Figure 5.9, then the Gamma curve segments G_3, G_4, $\cdots$, G_8 need to be adjusted. Finally, if the modified control point is interior to a cubic Bezier curve, then four consecutive Gamma curve segments need to be adjusted. Three of them are derived from the modified Bezier curve segment, and the other segment is derived from the adjacent Gamma curve segment which is closest to the modified control point because of the affected Overhauser curve segment. For example, if the control point P_5 in Figure 5.9 is modified, then the Gamma curve segments G_2, G_3, G_4, and G_5 need to be recomputed. However, if the control point P_6 is modified, then the Gamma curve segments G_3, G_4, G_5, and G_6 need to be recomputed.

5.7. Conversion to Bezier Curve

Since the convex hull property is very important in computer graphics, we would like to have the Gamma curve, which does not have the convex hull property, converted to the equivalent Bezier curve, which does. This is done by the equation given below:

$$P_{B\_G} = M_B^{-1} M_G P_v \tag{5.55}$$

where M_B^{-1} is the inverse of the matrix M_B

$$M_B^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{3} & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.56}$$

The four newly found control points contained in the matrix P_{B_G} are the vertices of the Bezier control polygon. After conversion, the Gamma curve is obviously contained within the convex hull of the control points specified by P_{B_G}. Also, the subdivision of the curve, as mentioned in Chapter Three, can be done easily now on the converted Gamma curve.

5.8. Gamma Surface

As discussed in Chapter One, the generalization of the Gamma curve to the Gamma surface can easily be done by forming the tensor product of two Gamma curves. The general equation for a Gamma surface patch is:

$$G_{i,j}(\gamma\,;\,u\,;\,s) = UM_{G_t} P_v (SM_{G_s})^T \tag{5.57}$$

where the parameter u gives a family of horizontal curves for the surface, while the parameter s gives a family of vertical curves for the surface. As a result of this, we have to use the matrix M_{G_t} for the horizontal curves and M_{G_s} for vertical curves. Both

matrices are derived from the matrix given in eq. (5.45). The matrix M_{G_i} requires the input of the i value of the ith horizontal patch for the computation of its $\beta = i \% 3$ value. The matrix M_{G_j}, on the other hand requires the j value of the jth vertical patch for the computation of its $\beta = j \% 3$ value. Obviously, both matrices require the input of the γ value. Note that there are three distinct matrices, with $\beta = 0$, $\beta = 1$ and $\beta = 2$ for each of M_{G_i} and M_{G_j}. However, it does not require six distinct matrices in memory. This fact is very useful in the implementation of the Gamma surface as mentioned in the next chapter. Also, we need a 6x6 matrix P_γ containing the mesh to derive a patch. Depending on the value of i and j, the mesh used within the matrix can be different, as in the case of the curve. When $i \% 3 = 0$, the first five control points in each row are used to provide the horizontal curve, when $i \% 3 = 1$, the middle four control points in each row are used, and when $i \% 3 = 2$, the last five control points in each row are used to derive the horizontal curve. Likewise, the parameter j works on the control points in each column, as i works on the control points in each row.

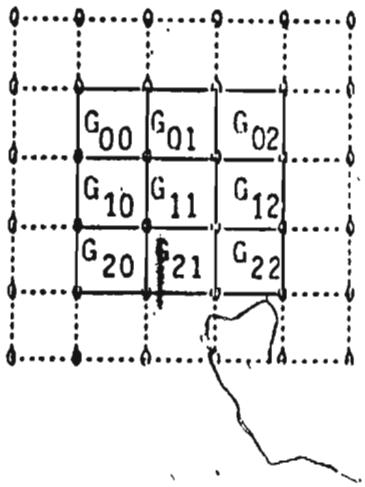


Figure 5.10 A bicubic Bezier surface patch, shown by the area enclosed by solid lines, is used to derive nine Gamma surface patches

Note that a bicubic Bezier surface patch is used to derive nine, 3x3, Gamma surface patches. This is shown in Figure 5.10, with the 6x6 matrix P_v given also. The inner 4x4 matrix of the 6x6 matrix P_v is required for the computation of the Bezier surface, and the surrounding control points in the 6x6 matrix P_v, are used by the Overhauser method to derive the nine Gamma surface patches. Note also that the derivation of the center patch G_{11} does not depend on the surrounding control points of the 6x6 matrix P_v.

As in the Gamma curve, the parameter γ gives the interpolated surface between the Bezier surface (when $\gamma = 0$), and the control points, or the Overhauser surface (when $\gamma = 1$). The four corner control points of the inner 4x4 matrix of the 6x6 matrix P_v are interpolated by the Gamma surface for any value of γ. The Gamma surface always maintains its smooth property if the Bezier surface and the Overhauser surface are C^1 continuous. A Bezier surface which is not C^1 continuous, will become C^1 continuous when $\gamma = 1$; that is, when the set of control points used to define the Bezier surface are interpolated. This is shown in Figure 5.11.

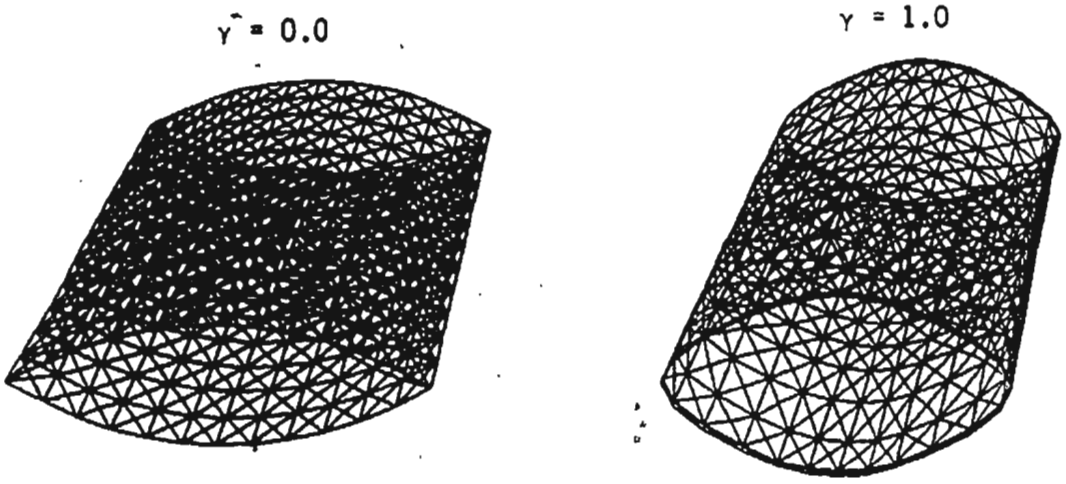


Figure 5.11 Gamma surface which does not have c' continuity at γ=0.0, will become c' continuous at γ=1.0

As in the case of the Gamma curve, the Gamma surface does not have the convex hull property. A way to obtain the convex hull property, similar to that of the Gamma curve, is to convert the Gamma surface to a Bezier surface to enforce the convex hull property. The formula to do this is as follows:

$$P_{B\_G} = (M_B^{-1}M_{G_s})P_v(M_B^{-1}M_{G_t})^T \tag{5.58}$$

where P_{B_G} is a mesh of 4x4 control points used to obtain the equivalent Bezier surface.

Locality is also achieved in the Gamma surface, as it is achieved in the Gamma curve. The number of surface patches which need to be adjusted varies according to the position of the modified control point. To see this, a four (2x2) Bezier surface patch layout is shown in Figure 5.12 with accompanying 9x9 control points.

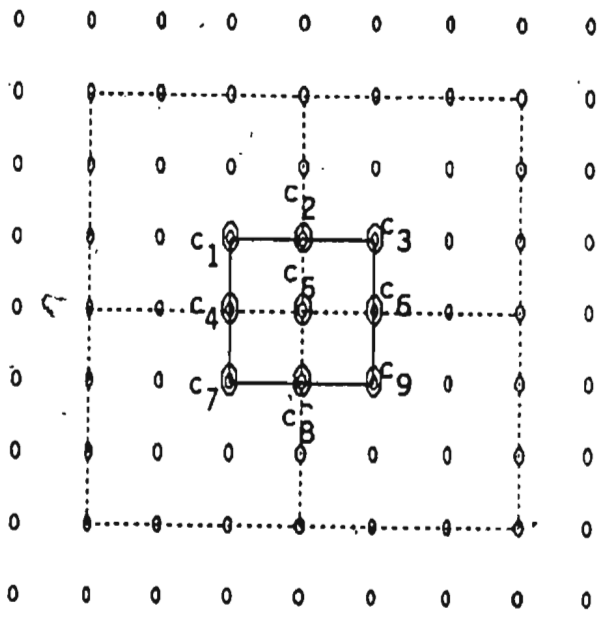


Figure 5.12 Layout of four bicubic Bezier surface patches with a focussed centre matrix of 3x3 control points numbered from c_1 to c_9

Here, our attention is focused on the 3x3 control points, numbered $c_1, c_2, \cdots, c_9$, which form the centre part of the four Bezier surface patches. Obviously, if the control point, c_5, which is at the corner of the four Bezier patches, is modified, then four (2x2) Bezier

patches need to be adjusted. This implies that thirty six (6x6) Gamma patches need to be modified. This is shown in Figure 5.13.

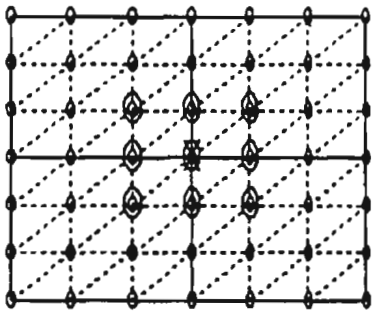


Figure 5.13 Thirty six Gamma patches need to be adjusted when the focussed point, c_5 (shown by a cross), is modified

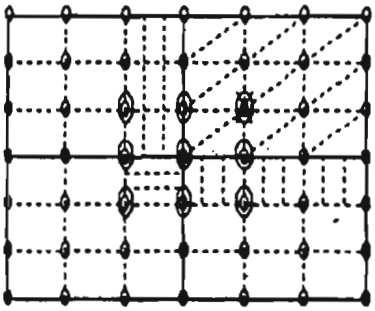


Figure 5.14 Modify c_3 causes adjustment of sixteen patches

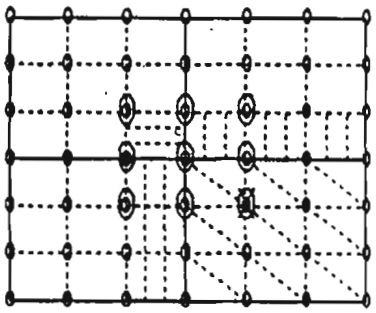


Figure 5.15 Modify c_9 causes adjustment of sixteen patches

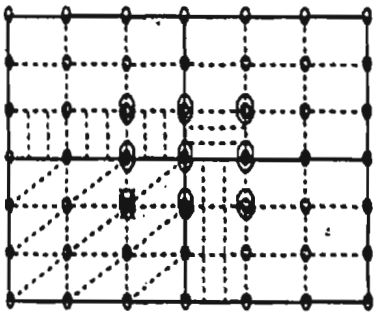


Figure 5.16 Modify c_7 causes adjustment of sixteeen patches

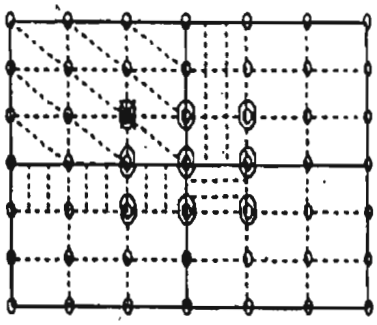


Figure 5.17 Modify c_1 causes adjustment of sixteeen patches

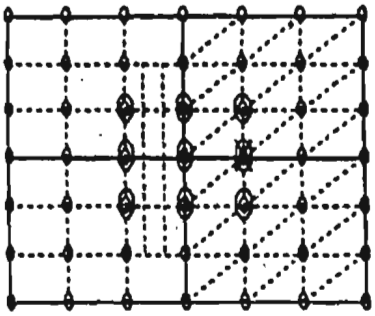


Figure 5.18 Modify c_6 causes adjustment of twenty two patches

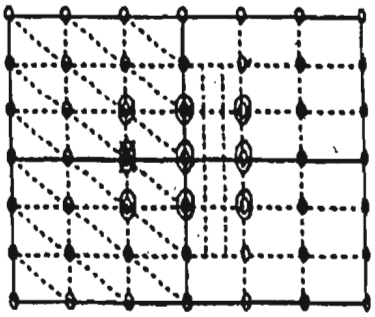


Figure 5.19 Modify c_4 causes adjustment of twenty two patches

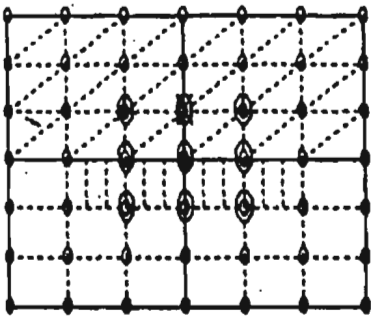


Figure 5.20 Modify c_2 causes adustment of twenty two patches

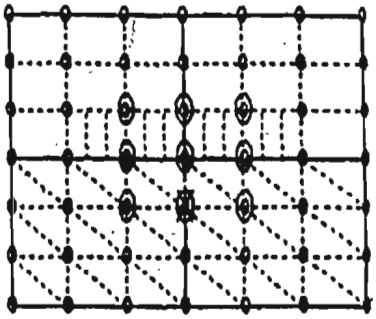


Figure 5.21 Modify c_8 causes adjustment of twenty two patches

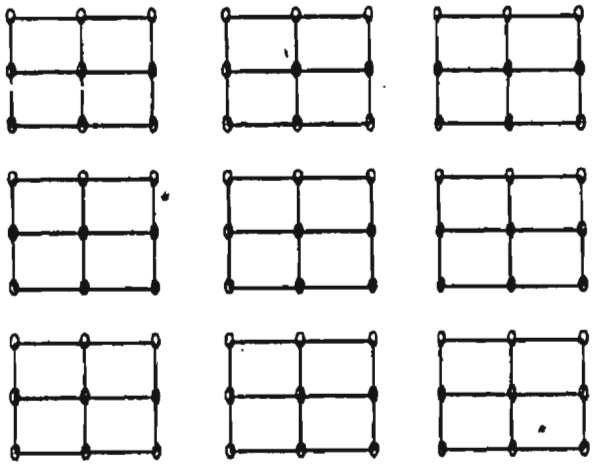


Figure 5.22 Pattern of the focused 3x3 matrix which is used to determine the way of adjusting the surface for a modified control point

If the modified control point is one of the corner points of the focused 3x3 center matrix, for example, c_3, then sixteen Gamma patches need to be recalculated, as shown by the shaded area in Figure 5.14. These are derived from the nine (3x3) Gamma patches, computed from the Bezier surface patch which is affected by the modification of the control point c_3 (shown by the area shaded by diagonal lines). In addition to that, three adjacent Gamma patches on each side of the modified Bezier surface patch closest to the modified point need to be recalculated (shown by the area shaded by the vertical lines). And finally, one Gamma patch that is diagonally closest to the modified point needs to be adjusted (shown by the area shaded by the horizontal lines). The last seven patches need to be recalculated because the Overhauser surface patches used to compute them are modified when the control point, c_3, is modified. Analogously, sixteen patches need to be recalculated in the same fashion for the modification of any of the remaining three corner control points. These are shown in Figure 5.15, Figure 5.16 and Figure 5.17.

Lastly, if one of the middle control points on any side of the 3x3 focussed matrix is modified, then twenty two Gamma patches need to be recalculated. The two Bezier surfaces sharing the modified control point, for example c_6 in Figure 5.12, have to be modified. This means that each of the nine Gamma patches derived from the two Bezier patches needs to be modified (shown by the area shaded by the diagonal lines in Figure 5.18). Also two adjacent patches closest to the modified point have to be modified (shown by the area shaded by vertical lines in Figure 5.18). The last four patches have to be recalculated because the four Overhauser surface patches used to compute them are modified. Analogously, twenty two patches need to be recalculated in the same fashion for the modification of any of the remaining three middle-side control points. These are shown in Figure 5.19, Figure 5.20 and Figure 5.21.

Now that we have worked up all the possible cases for the focused matrix, we can do the same for any other 3x3 matrix which is arranged in the fashion shown in Figure 5.22.

5.9. Summary

In this chapter, the derivation of the Gamma curve and the Gamma surface are discussed in detail. The Gamma technique is a generalization to the Bezier and the Overhauser techniques. The method proposes a way to mimic a Bezier curve or a Bezier surface to the control polygon or the net while maintaining their smooth appearance. By doing so, the convex hull property is lost. However, by converting the Gamma curve or surface to the equivalent Bezier, we can enforce the convex hull property. Locality can also be achieved with the Gamma method in the sense that by modifying a control point, only a few local curve segments or surface patches need to be recalculated. We would like to point out here that the parameter, γ, acts globally, as with the parameter β_2 in the Beta2-spline method. That is, by modifying the parameter γ, the whole Gamma curve or Gamma surface is also modified.

The matrices M_G, or M_{G_r} and M_{G_c} used to compute the Gamma curve or the Gamma surface respectively seem to be cumbersome. However, by separating them into three simple cases, the matrices are quite simple. This is examined in the next chapter where an implementation of the Gamma technique in an interactive environment on the SUN workstation is considered.

6. Implementation Of Gamma Technique

6.1. Introduction

Chapter Five discussed the formulation of the Gamma technique in detail. In this chapter, we will work on an implementation of the Gamma technique on the SUN workstation. A package is developed to draw three dimensional surfaces interactively using the Gamma technique.

6.2. Simplification

In order to have the processor execute the algorithm faster and more efficiently, we should simplify the algorithm as much as possible.

In the computer, we do not need to allocate a column of six control points or a mesh of 6x6 matrix of control points every time a new segment of a Gamma curve, or a new patch of a Gamma surface is drawn. The reason for this is that a column of six control points can provide the derivation of three consecutive Gamma curves, and a mesh of 6x6 of control points can provide nine patches of a Gamma surface; three patches in a row and three patches in a column. These facts are obvious from the arguments given in Chapter Five. Therefore, we can speed up the processing time by allocating a new column vector of six control points after we draw three consecutive curve segments from the current vector of control points, or in the case of a surface, by allocating a 6x6 mesh of control points after we draw nine patches of a Gamma surface from the current mesh of control points.

The matrix M_G, as specified in eq. (5.45) in the previous chapter, is quite cumbersome. This is because it combines all three distinct cases together, $\beta = 0$, $\beta = 1$ and

$\beta = 2$ in a single matrix. If we have a separate matrix for each case, then the matrix in each case can be simplified.

By substituting $\beta = 0$ into eq. (5.45), we get

$$M_{G\_0} = \begin{bmatrix} -\frac{\gamma}{2} & \frac{83\gamma-2}{54} & \frac{2-29\gamma}{18} & \frac{11\gamma-2}{18} & \frac{1-\gamma}{27} & 0 \\ \gamma & \frac{2-17\gamma}{6} & \frac{8\gamma-2}{3} & \frac{2-5\gamma}{6} & 0 & 0 \\ -\frac{\gamma}{2} & \gamma-1 & \frac{2-\gamma}{2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.1}$$

By substituting $\beta = 1$ into eq. (5.54), we get

$$M_{G\_1} = \begin{bmatrix} 0 & \frac{-25\gamma-2}{54} & \frac{2+25\gamma}{18} & \frac{-25\gamma-2}{18} & \frac{2+25\gamma}{54} & 0 \\ 0 & \frac{2+7\gamma}{9} & \frac{-13\gamma-2}{6} & 2\gamma & \frac{2-11\gamma}{18} & 0 \\ 0 & \frac{-8-\gamma}{18} & 0 & \frac{2+\gamma}{6} & \frac{1-\gamma}{9} & 0 \\ 0 & \frac{8-8\gamma}{27} & \frac{4+5\gamma}{9} & \frac{2-2\gamma}{9} & \frac{1-\gamma}{27} & 0 \end{bmatrix} \tag{6.2}$$

And by substituting $\beta = 2$ into eq. (5.45) we get

$$M_{G\_2} = \begin{bmatrix} 0 & \frac{\gamma-1}{27} & \frac{2-11\gamma}{18} & \frac{29\gamma-2}{18} & \frac{2-83\gamma}{54} & \frac{\gamma}{2} \\ 0 & \frac{1-\gamma}{9} & \gamma & \frac{-13\gamma-2}{6} & \frac{2+16\gamma}{9} & -\frac{\gamma}{2} \\ 0 & \frac{\gamma-1}{9} & \frac{-\gamma-2}{6} & 0 & \frac{8+\gamma}{18} & 0 \\ 0 & \frac{1-\gamma}{27} & \frac{2-2\gamma}{9} & \frac{4+5\gamma}{9} & \frac{8-8\gamma}{27} & 0 \end{bmatrix} \tag{6.3}$$

The matrices M_{G_0}, M_{G_1}, and M_{G_2} are simpler than those given by eq. (5.54). One can store these three matrices in the main memory for the computation of the Gamma curves or the Gamma surfaces. In the case of the curve, when $\beta = 0$, or $\beta = 1$, or $\beta = 2$, substituting the matrix M_{G_0}, or M_{G_1}, or M_{G_2} respectively for the matrix M_G in the eq. (5.46) will derive the corresponding curve segment. Analogously, substituting the matrix M_{G_0}, M_{G_1} or M_{G_2} for the matrices M_{G_i} and M_{G_k} in the eq. (5.57) appropriately will derive the corresponding (i, j) surface patch.

6.3. Map to Bezier

Since it is important that we preserve the convex hull property, we would like to convert the Gamma curve or the Gamma surface into Bezier form before we output them. The conversion of the Gamma curve to the Bezier curve is given in the eq. (5.55), as

$$P_{B\_G} = M_B^{-1} M_G P_v \tag{6.4}$$

where the matrix M_B^{-1} is the inverse of the Bezier matrix M_B

$$M_B^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{3} & 1 \\ 0 & \frac{1}{3} & \frac{2}{3} & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{6.5}$$

Now, by substituting the matrices M_{G_0}, M_{G_1}, and M_{G_2} into the matrix M_G of eq. (6.4), we will get three matrices as follows:

$$M_B^{-1} M_{G\_0} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -\frac{\gamma}{6} & \frac{\gamma+2}{3} & \frac{2-\gamma}{6} & 0 & 0 & 0 \\ 0 & \frac{8-5\gamma}{18} & \frac{5\gamma+4}{9} & \frac{2-5\gamma}{18} & 0 & 0 \\ 0 & \frac{8-8\gamma}{27} & \frac{5\gamma+4}{9} & \frac{2-2\gamma}{9} & \frac{1-\gamma}{27} & 0 \end{bmatrix} \tag{6.6}$$

$$M_B^{-1}M_{G\_1} = \begin{bmatrix} 0 & \frac{8-8\gamma}{27} & \frac{4+5\gamma}{9} & \frac{2-2\gamma}{9} & \frac{1-\gamma}{27} & 0 \\ 0 & \frac{8-17\gamma}{54} & \frac{4+5\gamma}{9} & \frac{2-\gamma}{6} & \frac{2-2\gamma}{27} & 0 \\ 0 & \frac{2-2\gamma}{27} & \frac{2-\gamma}{6} & \frac{4+5\gamma}{9} & \frac{8-17\gamma}{54} & 0 \\ 0 & \frac{1-\gamma}{27} & \frac{2-2\gamma}{9} & \frac{4+5\gamma}{9} & \frac{8-8\gamma}{27} & 0 \end{bmatrix} \tag{6.7}$$

and

$$M_B^{-1}M_{G\_2} = \begin{bmatrix} 0 & \frac{1-\gamma}{27} & \frac{2-2\gamma}{9} & \frac{4+5\gamma}{9} & \frac{8-8\gamma}{27} & 0 \\ 0 & 0 & \frac{2-5\gamma}{18} & \frac{4+5\gamma}{9} & \frac{8-5\gamma}{18} & 0 \\ 0 & 0 & 0 & \frac{2-\gamma}{6} & \frac{2+\gamma}{3} & -\frac{\gamma}{6} \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{6.8}$$

Now, by substituting the above three equations into the eq. (5.55) and the eq. (5.57) appropriately, the corresponding control polygon for the Bezier curve and the net for the Bezier surface can be derived. Note that the matrix $M_B^{-1}M_{G\_2}$ can easily be derived from the matrix $M_B^{-1}M_{G\_0}$ or vice versa. This is because the (i,j) (for $0 \leq i \leq 3$ and $0 \leq j \leq 5$) element in the matrix $M_B^{-1}M_{G\_2}$ is equivalent to the $(3-i,5-j)$ element of the matrix $M_B^{-1}M_{G\_0}$. Also, the rightmost three columns of the matrix $M_B^{-1}M_{G\_1}$ can be derived from the leftmost three columns of the matrix easily or vice versa. This is because the (i,j) (for $0 \leq i \leq 3$ and $3 \leq j \leq 5$) element is equivalent to $(3-i,5-j)$ element in the matrix itself. Also, notice that the above three matrices only depend on the value of γ. Therefore, the matrices need to be changed only if the value of γ is changed. By using these facts, we can speed up the computation time again.

6.4. Subdivision

In the implemented package, a user has the ability to control the number of squares to be formed per patch in any of the two directions along the surface patch. Each square is subdivided into four triangles for more accuracy of the surface shape. The algorithm is taken from reference [2]. After the Gamma surface is converted into an equivalent Bezier surface, it is subdivided along both the horizontal and the vertical direction as many times as the user wishes. Of course, the smaller the number of subdivisions, the more efficient is the computation of the surface. Although the efficiency is achieved, the resulting surface is not as smooth and precise as when a larger number of subdivisions is used. A large number of subdivisions will slow down the computation, but provide a finer mesh, and thus the resulting surface is smoother and more precise.

6.5. Algorithm

Given the above simplifications, the algorithm written in C to generate Gamma surfaces is produced below :

```
/* Draw nine surface patches for a given 6x6 matrix of control points */

draw_patch (gamma, vt)
double gamma;
double vt[6][6][3];

{
     int k, l;
     double bv[4][4][3];

     for (k = 0; k < 3; k++)
     {
          for (l = 0; l < 3; l++)
          {
               map_bezier (k, l, vt, bv);
               app_bezier_surface (bv, res_row, res_col);
```

```
            }
        }
    }


/* subdivide the patch along u and v for ech and gam times respectively and
   output the patch                                                        */

app_bezier_surface (w, ech, gam)
double w[4][4][3];
int ech, gam;

{
    double wlu[4][4][3], wru[4][4][3], wlv[4][4][3], wrv[4][4][3];

    if (ech > 0)
    {
        u_subdivide_bez_sur (w, wlu, wru);
        app_bezier_surface (wlu, ech-1, gam);
        app_bezier_surface (wru, ech-1, gam);
    }
    else if (gam > 0)
    {
        v_subdivide_bez_sur (w, wlv, wrv);
        app_bezier_surface (wlv, ech, gam-1);
        app_bezier_surface (wrv, ech, gam-1);
    }
    else
        polygon_app_bezier (w);

}


/* get the middle point of the control patch and output the 4 triangles */

polygon_app_bezier (w)
double w[4][4][3];

{
    double wa[3], w00[3], w03[3], w30[3], w33[3];
    int i;

    for (i = 0; i < 3; i++)
    {
        w00[i] = w[0][0][i];
        w03[i] = w[0][3][i];
```

```
            w30[i] = w[3][0][i];
            w33[i] = w[3][3][i];
        }
        for (i = 0; i < 3; i++)
            wa[i] = 0.25*(w00[i]+w03[i]+w30[i]+w33[i]);
        line_output_triangle (w00, w30, wa);
        line_output_triangle (w30, w33, wa);
        line_output_triangle (w33, w03, wa);
        line_output_triangle (w03, w00, wa);

}


/* subdivide along u */

u_subdivide_bez_sur (w, wlu, wru)
double w[4][4][3], wlu[4][4][3], wru[4][4][3];

{
    int i, k;

    for (i = 0; i < 3; i++)
    {
        for (k = 0; k < 4; k++)
        {
            wlu[0][k][i] = w[0][k][i];
            wlu[1][k][i] = 0.5*(w[0][k][i]+w[1][k][i]);
            wlu[2][k][i] = 0.25*(w[0][k][i]+w[2][k][i]) +
                           0.5*w[1][k][i];
            wlu[3][k][i] = 0.125*(w[0][k][i]+w[3][k][i]) +
                           0.375*(w[1][k][i]+w[2][k][i]);

            wru[0][k][i] = wlu[3][k][i];
            wru[1][k][i] = 0.25*(w[1][k][i]+w[3][k][i]) +
                           0.5*w[2][k][i];
            wru[2][k][i] = 0.5*(w[2][k][i]+w[3][k][i]);
            wru[3][k][i] = w[3][k][i];
        }
    }
}


/* subdivide along v */

v_subdivide_bez_sur (w, wlu, wru)
```

```
double w[4][4][3], wlu[4][4][3], wru[4][4][3];

{
    int i, k;

    for (i = 0; i < 3; i++)
    {
        for (k = 0; k < 4; k++)
        {
            wlu[k][0][i] = w[k][0][i];
            wlu[k][1][i] = 0.5*(w[k][0][i]+w[k][1][i]);
            wlu[k][2][i] = 0.25*(w[k][0][i]+w[k][2][i]) +
                           0.5*w[k][1][i];
            wlu[k][3][i] = 0.125*(w[k][0][i]+w[k][3][i]) +
                           0.375*(w[k][1][i]+w[k][2][i]);

            wru[k][0][i] = wlu[k][3][i];
            wru[k][1][i] = 0.25*(w[k][1][i]+w[k][3][i]) +
                           0.5*w[k][2][i];
            wru[k][2][i] = 0.5*(w[k][2][i]+w[k][3][i]);
            wru[k][3][i] = w[k][3][i];
        }
    }

}



/* output the triangle given the coordinates of the three corners */

line_output_triangle (w0, w1, w2)
double w0[3], w1[3], w2[3];

{
    move_abs_3 (w0[0], w0[1], w0[2]);
    line_abs_3 (w1[0], w1[1], w1[2]);
    line_abs_3 (w2[0], w2[1], w2[2]);
    line_abs_3 (w0[0], w0[1], w0[2]);

}


/* map the Gamma surface into the equivalent Bezier surface */

map_bezier (r, c, pv, w)
int r, c;
```

```
double pv[6][6][3], w[4][4][3];

{
        double pmt[6][4][3];


        switch (c)
        {
                case 0 : mat_pmt (pv, mg0, pmt);
                         break;
                case 1 : mat_pmt (pv, mg1, pmt);
                         break;
                case 2 : mat_pmt (pv, mg2, pmt);
                         break;
        }
        switch (r)
        {
                case 0 : mat_w (mg0, pmt, w);
                         break;
                case 1 : mat_w (mg1, pmt, w);
                         break;
                case 2 : mat_w (mg2, pmt, w);
                         break;
        }

}


/* Multiply the 6x6 matrix, which contains the control points, to
   the transpose of one of the 4x6 matrices: mg0, mg1 or mg2, which
   is passed in as the parameter mot, and the resulting matrix is
   returned through the parameter pmt */

mat_pmt (pv, mot, pmt)
double pv[6][6][3], mot[4][6], pmt[6][4][3];

{
        int i, j, k, l;

        for (l = 0; l < 3; l++)
        {
                for (i = 0; i < 6; i++)
                {
                        for (j = 0; j < 4; j++)
                        {
                                pmt[i][j][l] = 0.0;
                                for (k = 0; k < 6; k++)
```

```
                        {
                            pmt[i][j][l] += (pv[i][k][l] * mot[j][k]);
                        }
                    }
                }
            }

}


/* Multiply one of the matrices, mg0, mg1, or mg2, by the matrix, pmt,
   returned from the subroutine mat_pmt to derive the equivalent
   Bezier net, which is returned through the parameter w  */

mat_w (mo, pmt, w)
double *mo[4][6], pmt[6][4][3], w[4][4][3];

{
    int i, j, k, l;

    for (l = 0; l < 3; l++)
    {
        for (i = 0; i < 4; i++)
        {
            for (j = 0; j < 4; j++)
            {
                w[i][j][l] = 0.0;
                for (k = 0; k < 6; k++)
                {
                    w[i][j][l] += (mo[i][k] * pmt[k][j][l]);
                }
            }
        }
    }

}


/* initialize the 4x6 matrices mg0, mg1, and mg2 every time the value
   of the parameter Gamma is changed.  */

initialize_mg (gama)
double gama;


{
    initialize_mg0_2 (gama);
    initialize_mg1 (gama);
```

```
}


/* initialize the matrices mg0 and mg2. After mg0 is derived, mg2
   can be derived easily from the matrix mg0 */

initialize_mg0_2 (gama)
double gama;

{
      int i, j;

      for (i = 0; i < 4; i++)
            for (j = 2+i; j < 6; j++)
                  mg0[i][j] = 0.0;
      mg0[0][0] = mg0[2][0] = mg0[3][0] = 0.0;
      mg0[1][0] = -(gama / 6.0);
      mg0[0][1] = 1.0;
      mg0[1][1] = (gama + 2.0) / 3.0;
      mg0[2][1] = (8.0 - 5.0 * gama) / 18.0;
      mg0[3][1] = (8.0 - 8.0 * gama) / 27.0;
      mg0[1][2] = (2.0 - gama) / 6.0;
      mg0[2][2] = mg0[3][2] = (5.0 * gama + 4.0) / 9.0;
      mg0[2][3] = (2.0 - 5.0 * gama) / 18.0;
      mg0[3][3] = (2.0 - 2.0 * gama) / 9.0;
      mg0[3][4] = (1.0 - gama) / 27.0;

      for (i = 0; i < 4; i++)
            for (j = 0; j < 6; j++)
                  mg2[i][j] = mg0[3-i][5-j];

}


/* initialize the matrix mg1. After the first three columns are
   derived, the last three columns are easily derived from the
   first three */

initialize_mg1 (gama)
double gama;

{
      int i, j;

      mg1[0][0] = mg1[1][0] = mg1[2][0] = mg1[3][0] = 0.0;

      mg1[0][1] = (8.0 - 8.0 * gama) / 27.0;
      mg1[1][1] = (8.0 - 17.0 * gama) / 54.0;
```

```
    mg1[2][1] = (2.0 - 2.0 * gama) / 27.0;
    mg1[3][1] = (1.0 - gama) / 27.0;

    mg1[0][2] = mg1[1][2] = (4.0 + 5.0 * gama) / 9.0;
    mg1[2][2] = (2.0 - gama) / 6.0;
    mg1[3][2] = (2.0 - 2.0 * gama) / 9.0;

    for (i = 0; i < 4; i++)
        for (j = 3; j < 6; j++)
            mg1[i][j] = mg1[3-i][5-j];
}
```

In the algorithm given above, the matrices mg0, mg1, and mg2 stand for the matrices $M_B^{-1}M_{G\_0}$, $M_B^{-1}M_{G\_1}$, and $M_B^{-1}M_{G\_2}$ respectively, as mentioned in the previous section of this chapter. Every time the value of γ is changed, the three matrices have to be modified. The modification to the matrices mg0, and mg2 is done by the subroutine "initialize_mg0_2", while the modification of the matrix mg1 is done by the subroutine "initize_mg1". Notice how the simplification is done in both subroutines.

The algorithm for Bezier surface subdivision is taken from the paper[2]. Interested readers should refer to the paper for detailed discussion.

6.6. Description Of The Implemented Package

The implemented package runs on the SUN-Workstation. The package uses a keyboard and a mouse for input. The package will prompt the user for the next action to be taken as required. The user can select the "HELP" mode when in doubt. The display, on the display terminal, is divided into five regions (Figure 6.1): the feedback area (upper left), the current-value area (upper right), the display area (middle left), the menu switches area (lower right), and the menu potentiometers area (lower left). The particular menu switches displayed at any time are dependent on the current mode of the system.

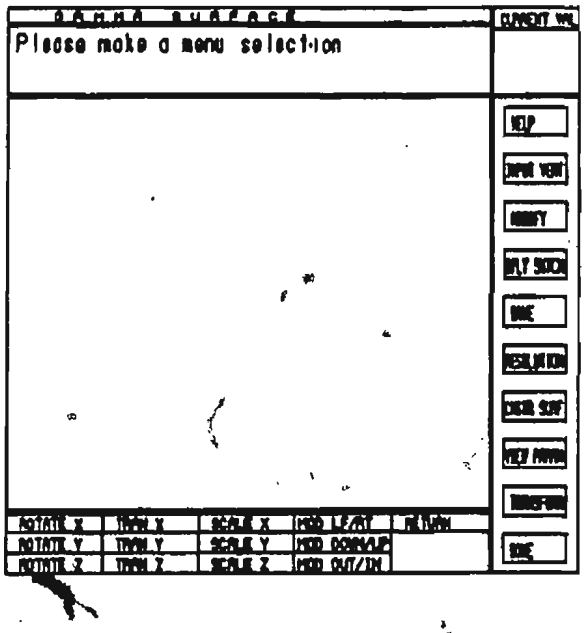


Figure 6.1 Main menu mode

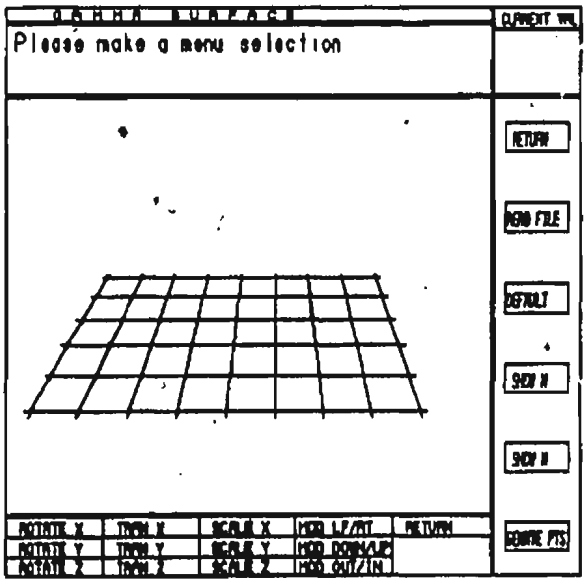


Figure 6.2 System supplied network of control vertices

The first step in the process is to specify the network of control points to be used, which can be accomplished by one of two techniques: an existing set of control points can be read from a file, or a new set can be generated interactively. If the latter alternative is selected, the number of control points in each direction in the network must be specified. This is done by prompting. Then, a default network of m control points by n control points is automatically generated as a regular grid in the xz plane with the same spacing in the x and z directions. Each control point is displayed as three mutually orthogonal line segments intersecting at the location of the point. Each control point is connected to its neighboring control points by dashed line segments called the control graph or the net (Figure 6.2).

To interactively modify the network of control points, the system is put in the modify mode, in which any point can be selected by pressing the left button on the mouse. Visual feedback is provided by blinking the selected control point. The value of the coordinates of the selected control point is displayed in the current-value area. Its position can now be modified using the three potentiometers marked MOD which control the amount of displacement in each dimension (Figure 6.3). This process is then repeated for all points that need to be moved to a satisfactory arrangement of control points (Figure 6.4). The value of the parameter γ can also be defined in the modify mode. If not defined, a default value of zero is assumed, which will produce a Bezier surface.

Having specified the network of control points, the next step is to create a surface for the network of control points. This is done by the construction mode. The user can go back to the modify mode to interactively refine the surface.

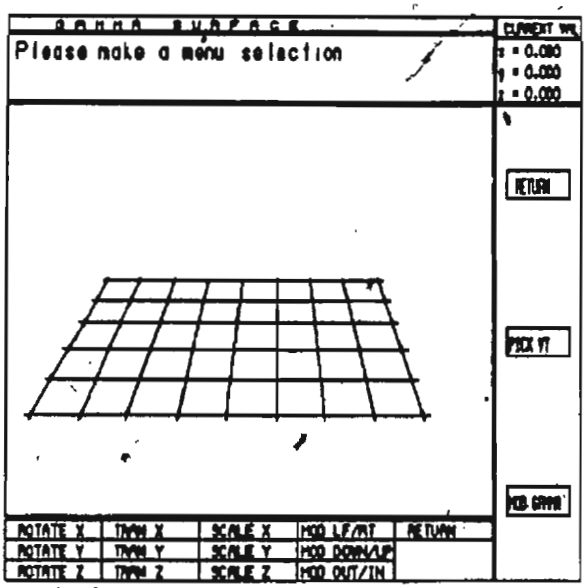


Figure 6.3 Modification of a selected control vertex

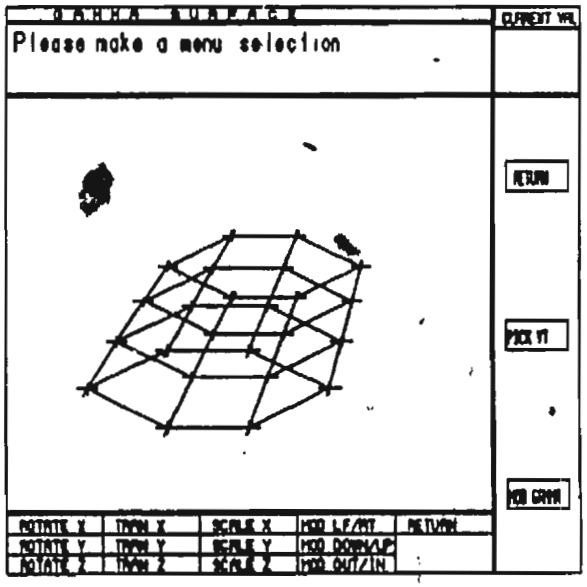


Figure 6.4 After complete modification of control points

The refinement is accomplished by moving one control point at a time as the movement of a single control point is guaranteed to affect only a local portion of the surface. If, however, the user changes the value of γ, then the complete surface is drawn. This is global control, so that the generated surface is pulled closer or further from the set of control points, depending on the modified value of γ.

As the amount of information being displayed is increased, it becomes more difficult to view the image. For this reason, the package can display only certain selected aspects of the image. This is accomplished through the use of a set of binary "display switches". The set of display switches, as shown in the menu display in Figure 6.5, enables the user to select any combination of control points, control graph, and surface to be displayed. For example, the control points, the control graph and the surface are shown in Figure 6.6, but only the surface is shown in Figure 6.7.

The default resolution of each surface patch is two squares in each direction. The user can specify this information in the resolution mode. The number of squares along each direction is two to the power of whatever the user specified. Figure 6.7 shows the picture with resolution 1, and Figure 6.8 shows the picture with resolution 2. The higher the resolution, the smoother and more precise is the surface. However, it is slower for the complete surface to be drawn.

The picture can be rotated, scaled, and translated. This is done by the left six menu potentiometers. Figure 6.9 shows the rotated picture of Figure 6.7.

The appendix of this thesis contains some of the pictures generated by the implemented package.

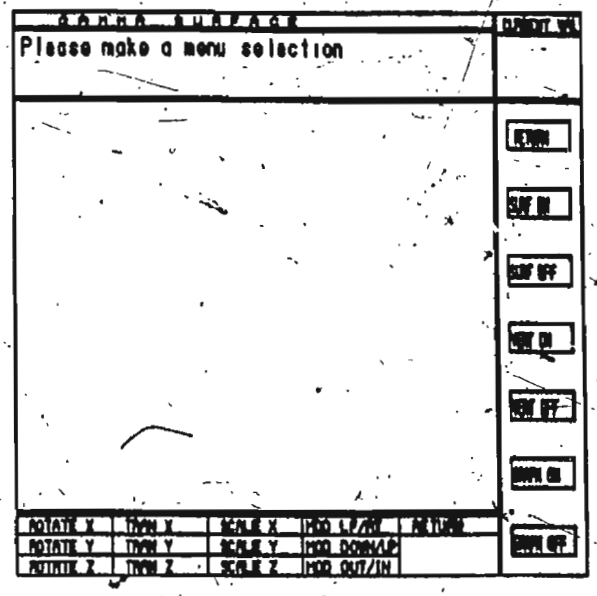


Figure 6.5 Display switches mode

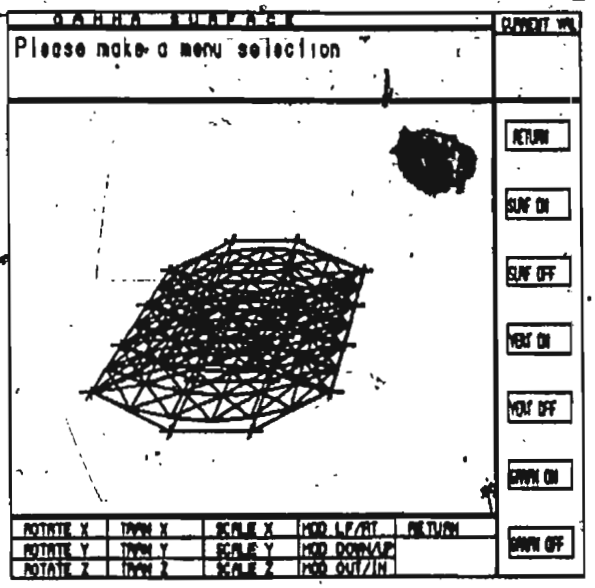


Figure 6.6 Surface, control points and control network are displayed at the same time

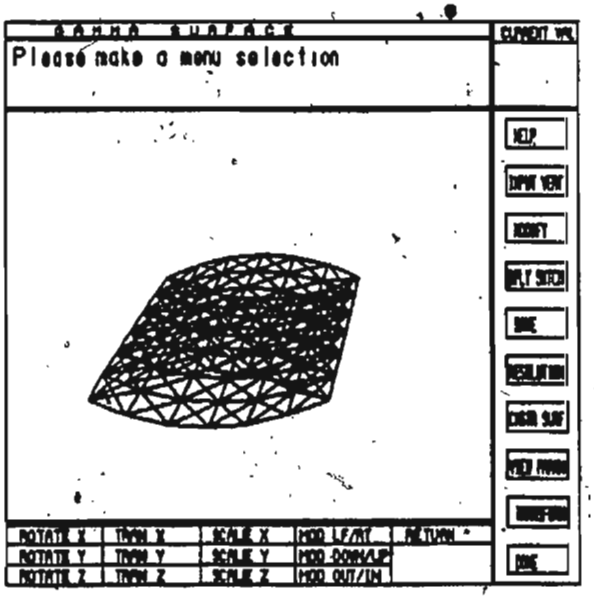


Figure 6.7 Only surface with resolution one is displayed

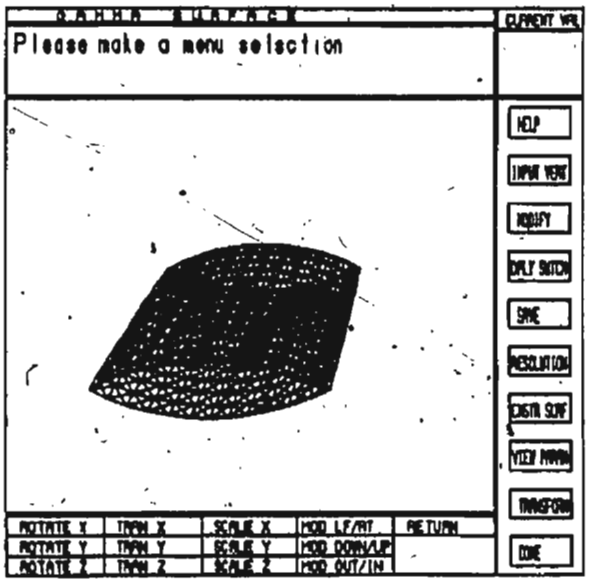


Figure 6.8 Surface with resolution two is displayed

6.7. Summary

An interactive 3D surface construction system is discussed in this chapter. Simplification to the algorithm to generate the Gamma surface is proposed in order to speed up the execution time in an interactive system.

The implemented package runs on a SUN-Workstation. The package provides the flexibility of both local and global control. Local control is achieved by moving one control point at a time, while global control is achieved by changing the value of the parameter γ. Other facilities include rotation, scaling, translation, and specifying any combination of control points, control graph, and surface to be displayed on the display area. At any moment the package will prompt for the next action. If an invalid response is entered, a warning or notification will be displayed at the feedback area. All these facilities are provided to make the package as user friendly as possible.

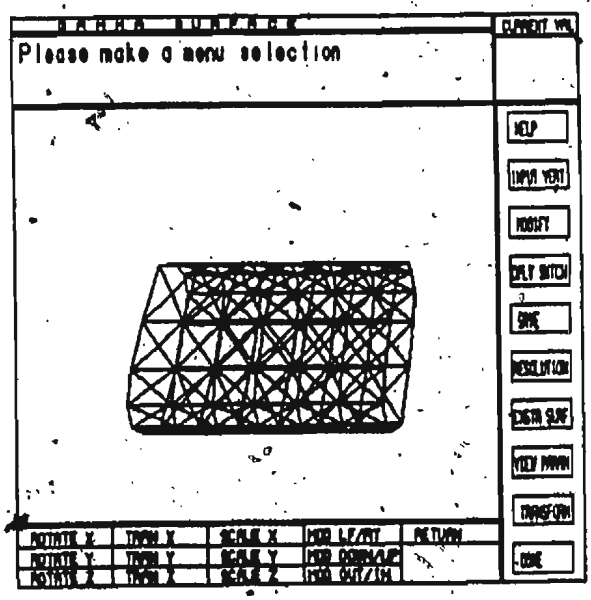


Figure 6.9 The rotated picture of Figure 6.7

7. Epilogue

7.1. Summary

Parametric curves and surfaces form an essential part of computer-aided design today. Schemes for defining these entities have evolved which employ a wide range of mathematical sophistication. The basic aim is to enable the designer to create curves and surfaces which behave as the designer wants them to behave. The Bezier method is a powerful tool for this purpose, and it is capable of representing most of the geometric entities of practical interest. It is not surprising that many researchers put enormous effort into enhancing and discovering the properties of the Bezier method.

The Bezier method, like other surface generation methods, is limited in that it may not accommodate the designer's concept of what the shape of the surface should look like. A shortcoming of the Bezier method is that the curve and the surface often bear little resemblance to the shape of the polygon and the net, respectively. The purpose of this thesis was to propose a simple method, the Gamma method, which can help the designer with this shortcoming of the Bezier method. The Gamma method is a generalization of the Bezier and the Overhauser methods. It can generate many curves or surfaces between the Bezier and the set of control points which possess better mimicking quality. This is done by varying the value of a parameter γ, from zero to one. Each distinct value of γ will produce a distinct curve or surface. When $\gamma = 0$, the corresponding curve or surface is just the Bezier curve or surface itself. As the value of γ increases towards one, the corresponding curve or surface is pulled towards the control points. When $\gamma = 1$, the curves or the surfaces generated interpolate the set of all control points, which is also the Overhauser curve or surface. For any value of γ, the generated curve or surface is

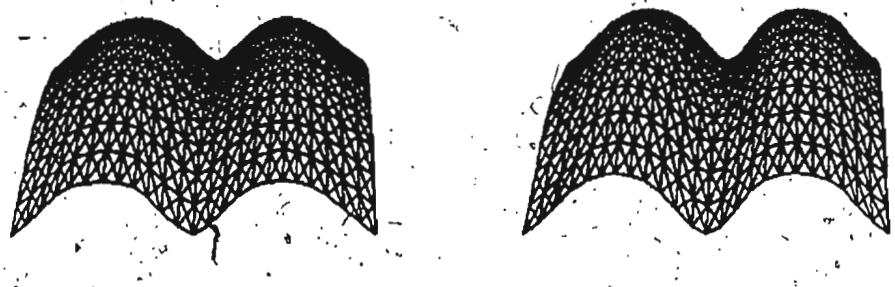
guaranteed to be smooth if the Bezier curve or surface is smooth. This is in contrast to the Beta2-spline, a generalization to the B-spline method, in which the generated curves or surfaces lose their smooth appearance as they are pulled closer to the set of control points, even though the B-spline curve or surface has C^2 continuity.

We believe the Gamma technique offers great flexibility to a general group of users. By setting the value of $\gamma = 1$, the user can interpolate the set of control points, which may be sample points from any physical phenomenon, such as temperatures taken from different places, depth of different locations of ocean floor, etcetera. Also it can be a useful Surface Modeling technique, which is the purpose of the proposal. The Gamma method presents a different idea in Surface Modeling.

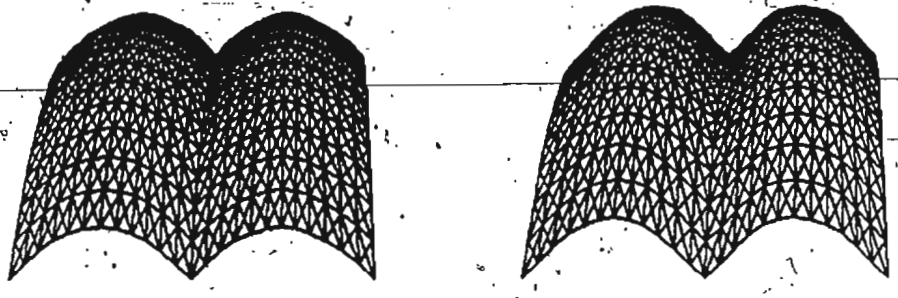
Finally, we hope the Gamma method is a pleasant technique to use whether by a sophisticated designer or by a novice.

Appendix

The following are sample pictures generated by the implemented package.

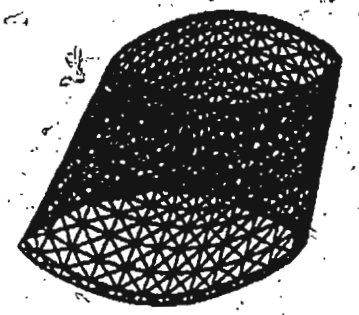


γ=0.67 γ=1.0

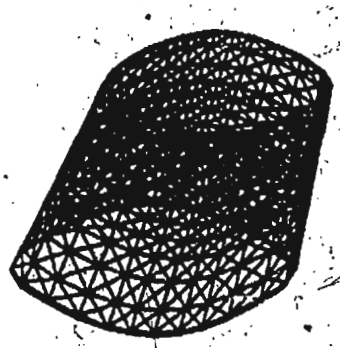


γ=0.0 γ=0.33

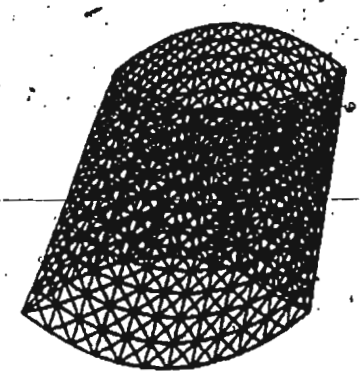
Figure A.1 Gamma surfaces with different parameter values



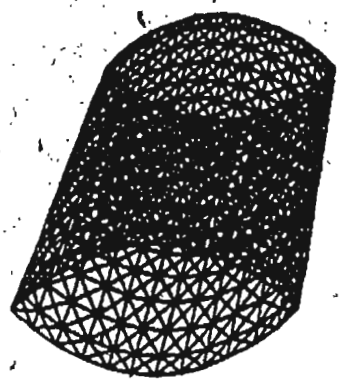
γ=0.67



γ=1.0



γ=0.0

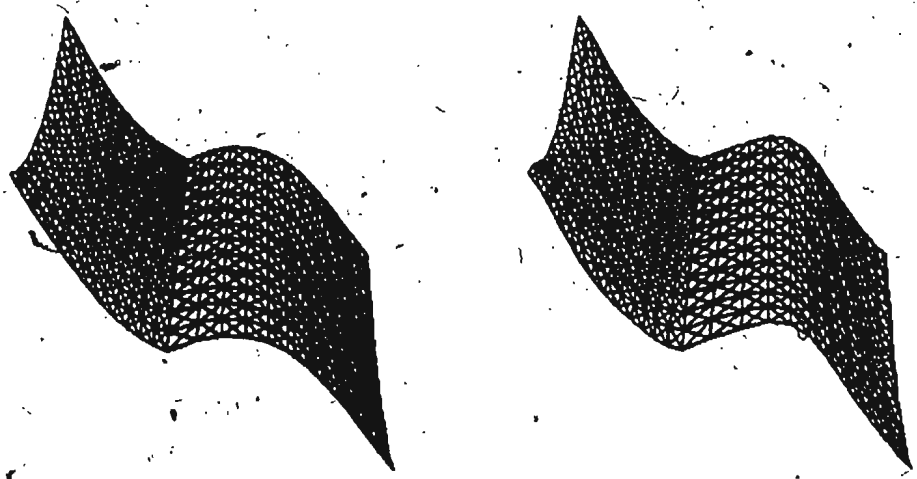


γ=0.33

Figure A.2 Gamma surfaces with different parameter values

γ=0.67

γ=1.0



γ=0.0

γ=0.33

Figure A.3 Gamma surfaces with different parameter values

References

1 **Barsky, Brian A.** *The Beta-spline : A Curve* and *Surface Representation* for *Computer Graphics* and *Computer Aided Geometric Design*, Computer Science Division, University of California, Berkeley, California (1983)

2 **Barsky, Brian A.** *The Beta 2-spline : A Special Case of the Beta-spline Curve* and *Surface Representation*, Computer Science Division, University of California, Berkeley, California (1983)

3 **Barsky, Brian A.** *Arbitrary Subdivision of Bezier Curves.*, Computer Science Division, University of California, Berkeley, California (1984)

4 **Bezier, Pierre E.** *Numerical control ; Mathematics* and *Applications*, John Welley and sons (1972)

5 **Blinn, J. F.** *Computer Display of Curved Surfaces.*, PhD Thesis, Computer Science Dept. University of Utah (1978)

6 **Brewer, J. A. and D. C. Anderson** "Visual Interaction with Overhauser Curves and Surfaces", *Computer Graphics*, *Association* for *Computing Machinery*, Vol 11, no 1 (summer 1977) pp 132-137

7 **Foley, James D. and Andries van Dam** *Fundamentals of Interactive Computer Graphics*, Addison Wesley (1984)

8 **Forrest, A. R.** *Curves* and *Surfaces* for *Computer Aided Design*, PhD Thesis, Cambridge University (1968)

9 **Forrest, A. R.** "On Coons and Other Methods for the Representation of Curved Surfaces", *Tutorial* and *Selected Reading in Interactive Computer Graphics* (1972) pp 381-399

10 **Forrest, A. R.** "Mathematical Principles for Curve and Surface Representation", *Proc Curved Surface Eng*, IPC Science and Technology Press, London (1972)

11 **Lane, Jeffrey M., Loren C. Carpenter, Turner Whitted, and James F. Blinn** "Scan Line Methods for Displaying Parametrically Defined Surfaces", *Communication of ACM*, Vol. 23, no. 1 (January 1980) pp 23-24

12 **Lane, Jeffrey M. and Richard F. Riesenfeld** "A Theoretical Development for the Computer Generation of Piecewise Polynomial Surface", *IEEE Transaction on Pattern Analysis* and *Machine Intelligence*, vol. PAMI-2

no. 1 (January 1980) pp 35-46

13 **Lee, Theodore M. P.** *Three Dimensional Curves* and *Surfaces* for *Rapid Computer Display*, PhD Thesis, Computer Science Dept. University of Harvard (1969)

14 **Overhauser, A. W.** "Analytic Definition of Curves and Surfaces by Parabolic Blending", *Scientific Laboratory Ford Motor Company* (1968)

15 **Riesenfeld, R. F.** *Application Of B-spline Approximation To Geometric Problems Of Computer-Aided Design*, PhD Thesis, Syracuse University (May 1973)